Creating a Simple Cloud Application

On installing Windows Azure SDK, a new project template group called *Cloud* is added in Visual Studio that includes a template, *Windows Azure Project* that we can use to create cloud applications. The Windows Azure Project template creates configuration files, Web roles, default web forms etc. automatically for us hence making our job of creating cloud application quite easier.

To test and run the cloud application in the local development environment, Visual Studio has to be run in *Elevated Administrator* mode.

To run Visual Studio in Elevated Administrator mode, select *Start* icon on the computer, right-click *Microsoft Visual Studio 2012* (or whatever version of Visual Studio you are using), and then click *Run as administrator* option.

Create a new cloud application by selecting *File* menu and select *New Project* option. From the *New Project* dialog that opens up, under *Installed Templates* heading, expand the *Visual* C# node and select *Cloud* node.

The Cloud node displays *Windows Azure Cloud Service* contained in it. On selecting the *Windows Azure Cloud Service* template, a default name, *WindowsAzure1**s* assigned to the new project. Select *Browse* button to specify the directory for keeping the project.

Though, we can assign any name to the project, but let's continue keeping the default name and select OK button to create the project as shown in Figure 2.1

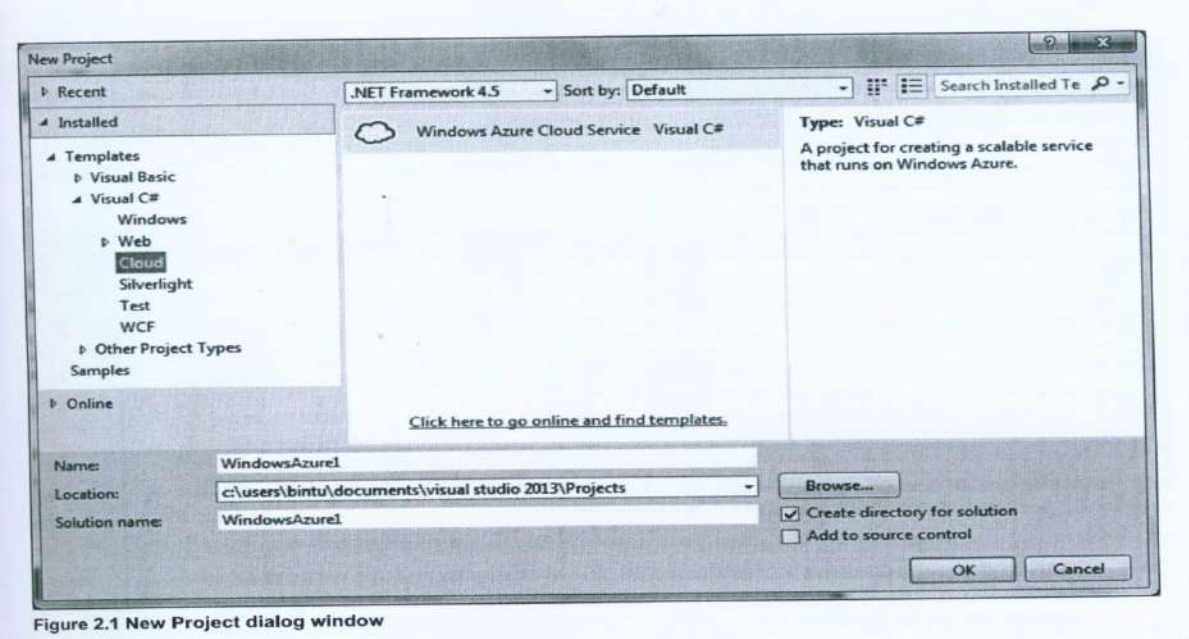


Figure 2.1 New Project dialog window

A New *Windows Azure Cloud Service* dialog is displayed prompting you to select Web Role for the cloud project shown in Figure 2.2. A Web Role is a web application that interacts with the user. So, one of the displayed Web Role projects needs to be added to our cloud application.

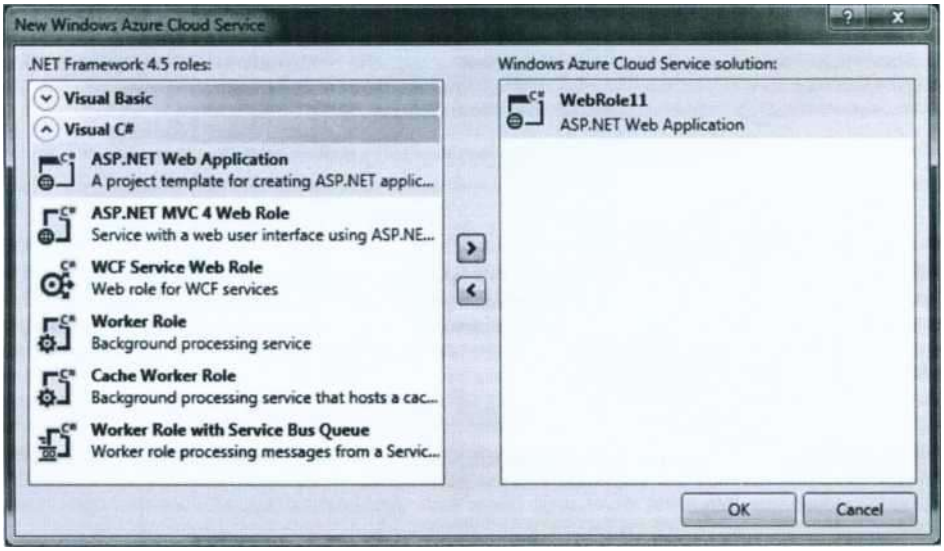


Figure 2.2 Dialog to add roles to the Windows Azure Cloud Service

The figure displays the following list of the Windows Azure Project options:

ASP.NET Web Application - Creates an ASP.NET Web application that supports cloud.

- **ASP.NET MVC4 Web Role** - Creates a project similar to the ASP.NET Web Application template that supports the MVC4 framework.

WCF Service Web Role - Creates a WCF (Windows Communication Foundation) project.

Worker Role - Creates a class library project that is used for building a background processing service.

- **Cache Worker Role** - Creates a role that provides a dedicated cache to our application.

Worker Role with Service Bus Queue - Creates a service bus queue that provides message queuing functionality to communicate with the worker process.

At the moment, we are interested in creating a simple web application for user interface, so select *ASP.NET Web Application* project from the Visual C# list and

select “>” button to add it to the *Windows Azure Cloud Service solution* window. The selected web application appears in the *Windows Azure Cloud Service solution* window with the default name, *WebRolell* which can be changed by selecting the *Pencil* icon that appears adjacent to it. Let's go ahead, keeping the default name of the Web Role project and select *OK* button to move further. The next dialog prompts to select the template for the ASP.NET Web Application as shown in Figure 2.3. Below given is the list of project templates:

Empty Template - Creates the project with minimum references and resources required to run it.

Web Forms Template - Helpful in creating user interfaces and implements data access from the back end.

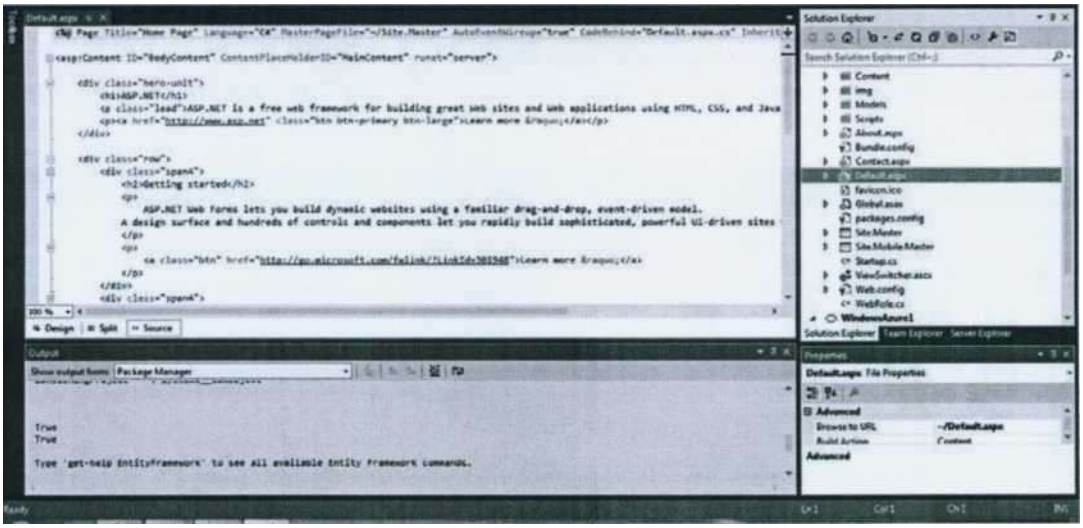
MVC Template - Helpful in creating MVC based project that separates the logic from the user interface. It divides an application into three parts, Model, View and Controller.

Web API Template - Creates a project that provides a sample of Web Service that is based on the Web API. Useful in creating HTTP services.

SPA (Single Page Application) Template - Used to develop a single page application. The template provides resources that are required for developing a single page application.

- **Facebook Application.** The template includes a library that provides all the necessary functionalities involved in building a Facebook Application. It creates a sample application designed to run in the Facebook web site.
- **Mobile Template** - Creates an application for mobiles based on the ASP.NET MVC. Useful to create applications that run on both browser and mobile clients.

Because we want to create a simple Hello World application, let us select Web Forms template and click Create Project button.



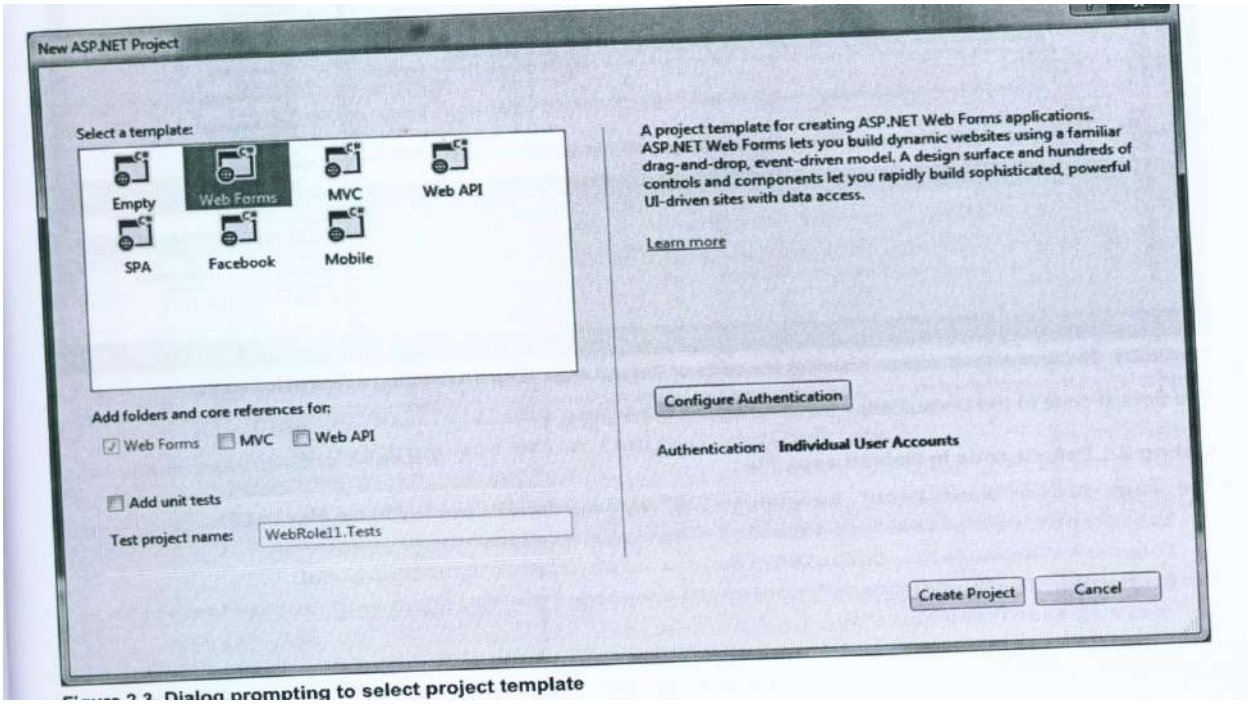


Figure 2-3. Dialog prompting to select project template

WindowsAzurel and *WebRolell* where:

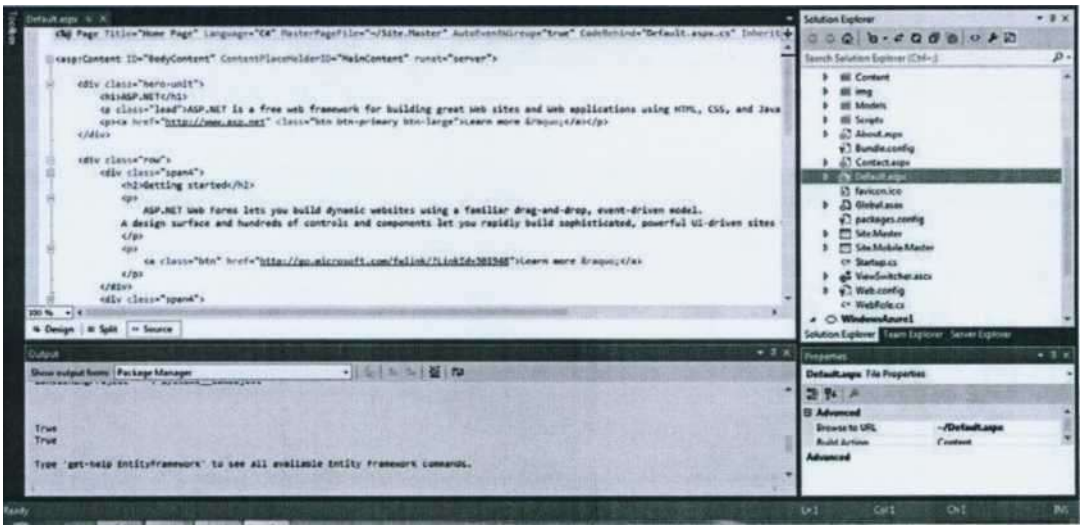


Fig. 2.4 The first default screen showing the code of Default.aspx form on ceating a new solution

- The *WindowsAzurel* project is the new cloud services project containing configuration and metadata of the Windows Azure cloud service along with the reference to the web role project. The project includes a *Roles* folder where references to all the roles in the cloud service are kept.

The *WebRolell* project is the ASP.NET web application that interacts with the user. Like a traditional web application, this project consists of one or more web forms where different controls can be dragged and dropped from the *ToolBox* to create user interface.

The default code in the Default.aspx file is as shown in Listing 2.1

Listing 2.1 Default code in Default.aspx file

<%0 Page Title="Home Page" Language="C#" MasterPageFile="~/Site.Master" AutoEventWireup="true" CodeBehind="Default.aspx.cs" Inherits="WebRolell._Default" %>
<asp:Content ID="BodyContent" ContentPlaceHolderID="MainContent" runat="server">
<hl>ASP.NET</hl>
ASP.NET is a free web framework for building great Web sites and Web applications using HTML, CSS, and JavaScript.
Learn more Sraquo;

<h2>Getting started</h2>

ASP.NET Web Forms lets you build dynamic websites using a familiar drag-and-drop, event-driven model.
A design surface and hundreds of controls and components let you rapidly build sophisticated, powerful Ul-driven sites with data access.

Learn more »

<h2>Get more libraries</h2>
<P>
NuGet is a free Visual Studio extension that makes it easy to add, remove, and update libraries and tools in Visual Studio projects.

<P>
Learn more »

<h2>Web Hosting</h2>
<P>
You can easily find a web hosting company that offers the right mix of features and price for your applications.

<P>
Learn more »

</asp:Content>

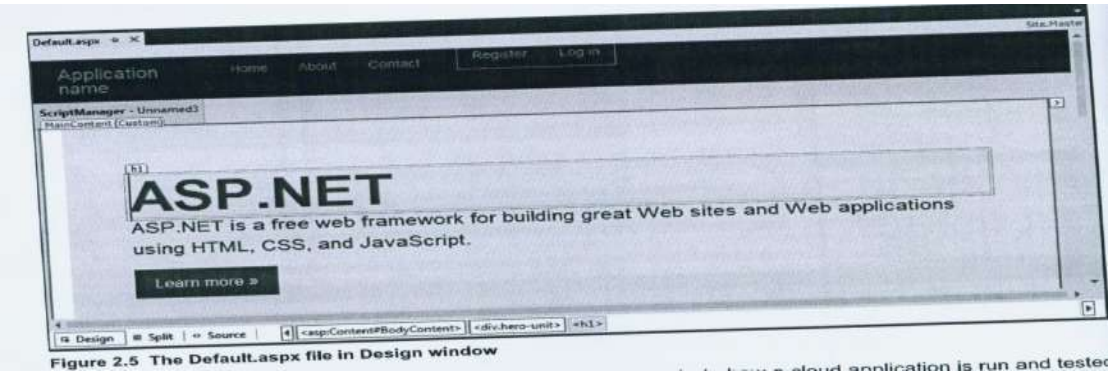


Figure 2.5 The Default.aspx file in Design window

The purpose of creating this simple cloud application is to demonstrate how a cloud application is run and tested in the local cloud environment and on actual Windows Azure but not to exploit different storage services provided by Windows Azure. Hence, we will make the web form, *Default.aspx* to just display a simple text message, *Welcome to Cloud Computing.* Modify the code to appear as shown in listing 2.2.

Listing 2.2 Code in Default.aspx file

<%0 Page Title="Home Page" Language="C#" AutoEventWireup="true"

CodeBehind="Default.aspx.cs" Inherits="WebRolell._Default" %>

<h2>

Welcome to Cloud Computing </h2>

After making the changes, save the file. To run and debug the Application, from the Menu Bar, select Debug- *>Start Without Debugging* option. Our application will run

in a simulated cloud environment as the *Compute Emulator* displaying the output, "Welcome to Cloud Computing" as shown in Figure 2.6 .

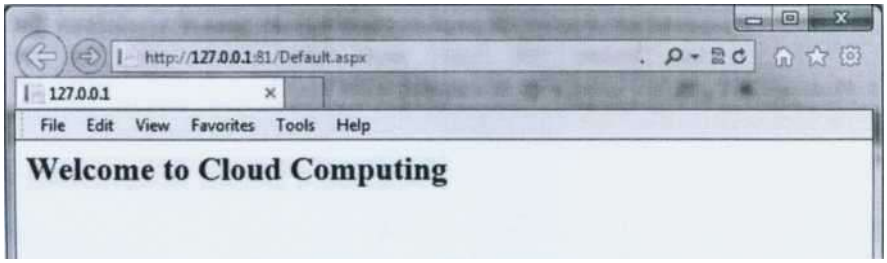


Figure 2.6 Application running in the local simulated environment

Fig. 2.6 Application running in the local simulated environment

- Since, we are talking about running applications, it is necessary for you to know that the two main services tha' Windows Azure runtime environment provides are as given below known Compute - Refers to the computational tasks, running and managing roles, their instances, allocating space for VM and so on

- Storage - Refers to the three main storage services: Table service, Blob service and Queue service. Besides these three Windows Azure Storage Services, data can be stored in SQL Azure also

Compute Emulator

The *Compute Emulator* is a simulation of the Windows Azure fabric that simulates running role instances on our machine the way it will run on Windows Azure. It provides an environment to build, debug, and test applications locally before deploying them to the cloud.

In the cloud, our application runs on different virtual and physical machines. Every instance of the role is hosted in a separate VM. In the local simulation environment, for every virtual machine launched in the cloud, the *Compute Emulator* launches a *local process* to host our application.

Remember, *Compute Emulator* simulates cloud environment and is not the actual cloud. It is just a process on our local machine hence our experience of executing the application in *Compute Emulator* may be different from the one when deployed in the cloud.

For example, there may be high latency time when our application accesses Windows Azure storage services because of network delay. Whereas, when deployed, the application access the Windows storage services comparatively faster as it exists in the same data center. In short, you might feel a vast difference in the performance of the application when run in *Compute Emulator* and in actual cloud.

While running an application in the local simulation environment, the Visual Studio communicates with the *Compute Emulator* to run the project.

To run and test an application in the Compute Emulator, Visual Studio should run in *Elevated Administrator* mode. If the Visual Studio is not running as an administrator, then on running the project, an error appears, The *Compute Emulator Must Be Run Elevated. Please Restart Visual Studio In Elevated Administrator Mode In Order To Run The Project.*

In the Notification Area of the Windows Taskbar when we click the *Windows Azure Emulator* icon, a message is displayed, *Compute Emulator is started* and *Storage Emulator is started* as shown in Figure 2.7. The message confirms that our application is running in a simulated cloud environment. If we don’t find Windows Azure Emulator icon in the Notification Area, select *Start->Ail Programs->Windows Azure -> Emulators* option, and select Windows Azure *Compute Emulator* option from the list of displayed Emulators.

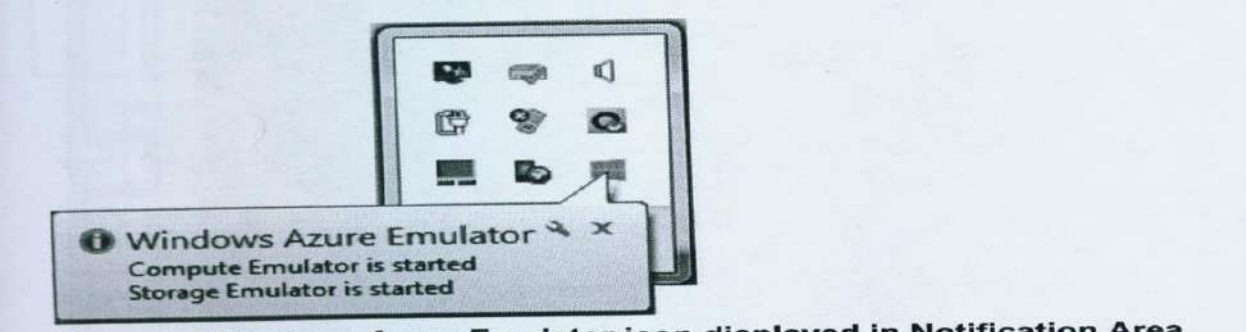


Figure 2.7 Windows Azure Emulator icon displayed in Notification Area

Note: The Storage Emulator provides a local storage services similar to the actual storage services. Table, Blob and Queue services provided by Windows Azure. That is, it provides a local environment to test and run the applications related to storage services. It provides a UI through which, we can not only view the status of the local storage services but also can start, stop, and reset them.

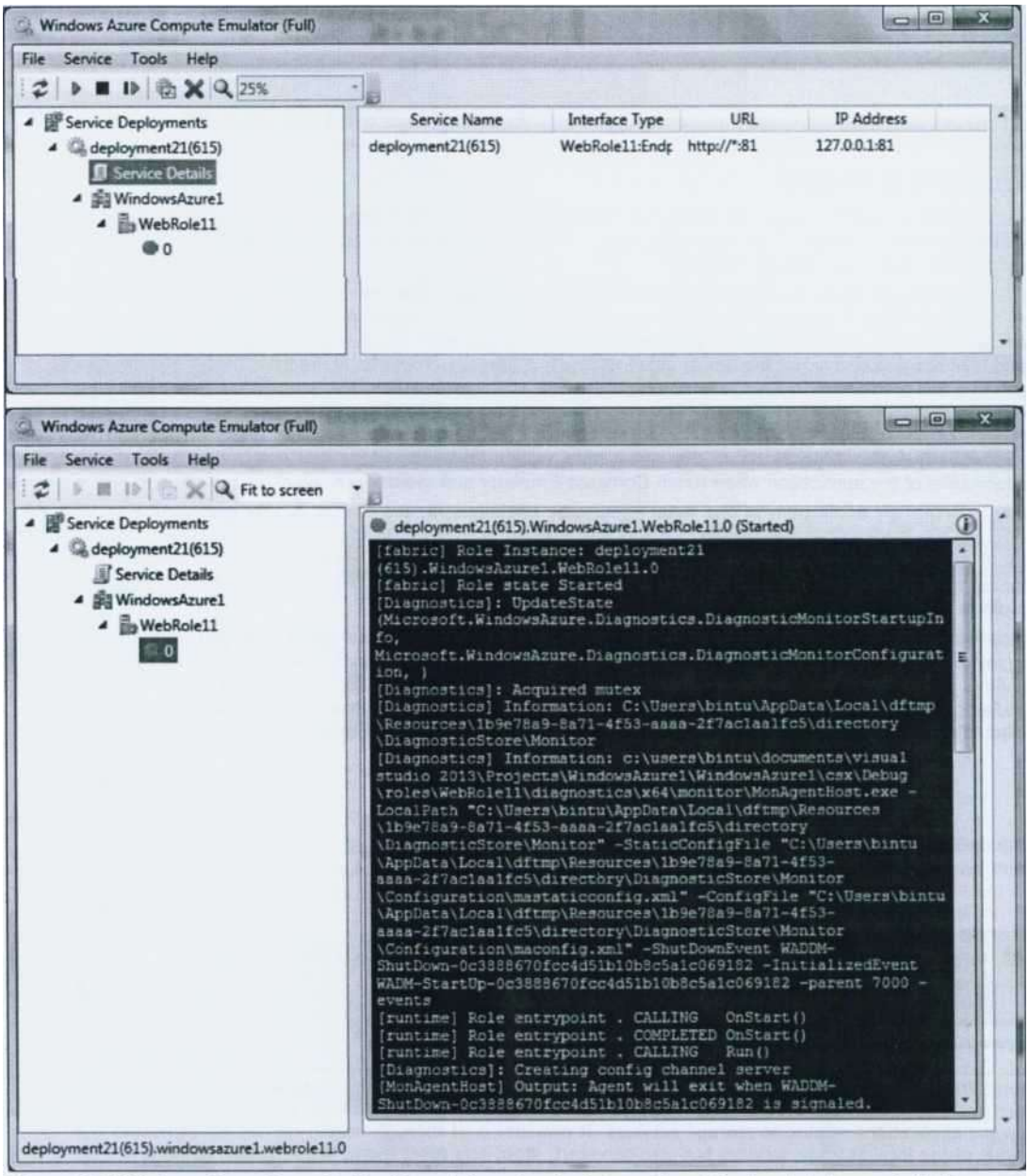


Figure 2.8(a) Windows Azure Compute Emulator window showing information of the local deployment (b) Messages

Configuring an Application

To deploy an application, certain configuration settings need to be provided to the Windows Azure Fabric that helps it in running and managing the service. Through configuration settings, the information that is provided to Fabric includes the disk space and other resources that the service may require, the ports that service will listen for requests, number of roles that the service has and the number of instances that each role has, database connection string if the service is using some storage service, and much more. That is, through configuration settings, Windows Azure Fabric will have a complete know how of the running service. The configuration information is provided through service definition and service configuration files. Let's begin with Service Definition file.

Understanding Service Definition file

The Service Definition file, *ServiceDefinition.csdef* defines the information like input endpoints for the service and the information of different roles available in the service.

It also defines the disk space limit and other resources that the service can access and the ports that the service will use to listen for the requests. The settings defined in this file are used by the Fabric Controller to manage the service.

The information in service definition file cannot be changed dynamically i.e. to apply the changes and to bring them into effect, the application needs to be rebuild and its service package file has to be uploaded again. We will see to this later in this chapter.

The default content in the Service Definition file in the project, WindowsAzurel that we created above will be as given below:

Listing 2.3 Code in Service Definition file, ServiceDefinition.csdef

```
<?xml version="l.0" encoding="utf-8"?>
<ServiceDefinition name="WindowsAzurel"
xmlns="http://schemas.microsoft.com/ServiceHosting/2008/10/ServiceDefinition"
schemaVersion=''2013-03.2 . 0">
<WebRole name="WebRolell" vmsize="Small">   #1
<Sites>
<Site name="Web">#2
<Bindings>
<Binding name="Endpointl" endpointName="Endpointl" />       #3
</Bindings>
</Site>
</Sites>
<Endpoints>
<InputEndpoint name="Endpointl" protocol="http" port="80" /> #4 </Endpoints>
<Imports>
CImport moduleName="Diagnostics" />
</Imports>
</WebRole>
</ServiceDefinition>
```

The *name* attribute of *ServiceDefinition* element provides a name to the service, *WindowsAzureI* which is used by the Compute Emulator and the Windows Azure Fabric to define a contract for a service.

Through the service definition file, the size of the VM required for installing and running the web role of the service is defined.

Specifying VM Size

Recall, each physical server is divided into multiple VMs where a VM is a software implementation that emulates a computing environment where every instance of the role whether web role or worker role of the service is installed and run.

Remember, each instance of role is installed into a separate VM. The statement #1 informs that our Cloud Service contains only a single web role, called *WebRoleI* and it needs a VM of *small* size. The table 2.1 shows the size and configuration of the VMs if they were real machines.

Size	CPU	RAM	Hard Disk
ExtraSmall	Shared	768 MB	1 TB
Small	One core	1.75 GB	2 TB
Medium	Two cores	3.5 GB RAM	4 TB
Large	Four cores	7.0 GB RAM	8 TB
Extra Large	Eight cores	14 GB RAM	16 TB

The above table demonstrates how the RAM size, Hard Disk capacity and cores in the CPU increase with the Increase in size of VM. The performance of a service can be increased either by increasing the number of role Instances or by increasing the size of VM. The option of increasing the size of virtual machine is quite cheaper than having multiple role instances. But we need to experiment the two options with our application and check its performance before choosing the one.

The *Site* element in #2 (in Listing 2.3) can be used for configuring multiple sites on the same Web Role. At the moment it defines a single site by a default name, *Web.* The Binding element in #3 is used for identifying an endpoint. Let's have an idea of what an Endpoint means.

Defining Endpoint

Endpoint is used to define the port and the protocol that a service will use to listen for requests. Depending on the type of interface supported by the service, several endpoints can be defined in it. Each endpoint is defined within the *<Endpoints>* element. The *name*

and *endpointName* attributes of the *Binding* element in (code listing 2.3) in statement #3 identifies that there is a single endpoint in the service and that is defined by name, *Endpointl.*

The statement #4, defines the endpoint of the service by name, *Endpointl* that configures the web role, *WebRolel* to listen to *HTTP* protocol on port 80. Windows Azure allows web roles to receive incoming HTTP or HTTPS messages preferably via port 80 and port 443 respectively (though we can use other ports too). The idea behind defining the protocol and port via Endpoint in Service Definition file is that all incoming requests to the web role from the protocols and ports other than the ones defined in the service definition file are filtered out by the firewalls and load balancers hence making it more secure.

As said earlier, we can define several endpoints within *<Endpoints>* element. For example, to configure the web role to use HTTPS, another endpoint can be defined in the <Endpoints> section as shown below

<InputEndpoint name=”Endpoint2'' protocol="https" port="443"/>

Above statement configures the web role to listen to HTTPS protocol on port 443.

Where web roles support only HTTP protocol, the worker roles can be configured to listen to TCP protocol which is comparatively better in performance over HTTP protocol. For example, the following statement configures a worker role to listen to TCP protocol on port 99:

<InputEndpoint name="WorkerEndpoint" protocol=”tcp" port="99" />

There are a few elements that we don’t find in above Service Definition file. A small description of them is as given below:

Internal Endpoint

The web roles that are meant to only communicate with other roles of the service and are not available outside the Windows Azure fabric are configured through internal endpoint. The internal endpoint of a role is configured by setting the *name, port,* and *protocol* attributes of the *<IntemalEndpoint>* as shown in below statement:

cinternalEndpoint name="InternalWebRolel" protocol="http''/>

Above statement defines an internal endpoint by name, *IntemalWebRolel* that configures the web role to listen to *HTTP* protocol for internal communication with other roles of the service. The ports for the internal endpoints are dynamically assigned and cannot be manually set by us. The following statement defines an internal endpoint called *WorkerEndpoint* that uses *TCP* protocol:

cinternalEndpoint name="WorkerEndpoint” protocol="tcp" />

Note: Though the Web Role project in above application is configured to use port 80 through the Service Definition file, notice that while deploying the application in local simulated environment, it uses the port 81 (refer Figure 2.8(a). It is so, because the IIS server uses the port 80 by default and when the *Development Fabric* finds the port 80 already in use, it

assigns the next available port to the web role. Hence, the next available port, 81 is assigned to the web role.

Trust Level

Windows Azure supports two levels of trust: full trust and partial trust. The partial trust level provides restricted access to the roles i.e. they won't be able to access native libraries and resources whereas full trust is the default level and does not applies any restriction to the web roles in performing their operations. So, the partial trust level implements security and to configure it, we set the enableNativeExecution attribute on our role to false :

<WebRole name="WebRolel" enableNativeExecution="false">

After understanding the elements that make up the Service Definition file, let's have a look at another important file that helps in implementing configuration settings of a service, Service Configuration file.

Understanding Service Configuration File

The Service Definition file that we discussed above defines just the names of the settings and no values. The values for the settings defined in Service Definition file are specified in the Service Configuration file. Configuration settings like number of role instances, storage account name (database connection string), service endpoints etc. are specified in this file. The settings in this file can be changed at runtime i.e. to bring the changes made in the settings into effect, we don't have to rebuild the application or re-upload the service package file. The ability to change the settings in this file dynamically makes it possible to deploy an application between Staging and Production environments. We will be discussing about the Staging and Production environments in detail later in this chapter. For the time being, it's enough to know that an application is deployed in Staging environment for the purpose of testing the code in the cloud. After testing, the deployed application is upgraded to Production environment for releasing the tested code for the clients or consumers. The ability to change the settings dynamically also makes it easy to switch to different storage account without making any changes in the source code of the application.

There are two configuration files, ServiceConfiguration.Cloud.csfg and ServiceConfiguration.Local.cscfg where former is used for storing settings for the local environment and the later is for storing settings for the cloud environment respectively. Initially, the content of both the files are same and appear as shown in Listing 2.4.

Listing 2.4 Code in configuration files,ServiceConfiguration.Cloud.csfg and ServiceConfiguration.Local.csfg

<?xml version="l.0" encoding="utf-8"?>

<ServiceConfiguration serviceName="WindowsAzurel"

xmlns=”http://schemas.microsoft.com/ServiceHosting/2008/10/ServiceConfiguration" osFamily="3" osVersion="*" schemaVersion="2013-03.2.0">

<Role name="WebRolell">

clnstances count=”l" />

<ConfigurationSettings>

<Setting name="Microsoft.WindowsAzure.Plugins.Diagnostics.ConnectionString" value="UseDevelopmentStorage=true" />

</ConfigurationSettings>

</Role>

</ServiceConfiguration>

We can see that the first element in the Service Configuration file is the *<Role>* element. It defines the number of Instances of the web role. Let's know more

Defining the number of role instances

ror reach web role defined in the Service Definition file, there is a <Role> element defined in the Service Configuration file. Since, our Cloud Service project contains only a single web role, only one <Role> element appears in the Sen/ice Configuration file that informs that the current application consists of a single Role by name, *WebRolel.* The element contains two main sections, *Instances* and *ConfigurationSettings.* The Instances tection informs the number of instances of the web role. The *count* attribute of the <lnstances> element is set to value 1 to indicate that we wish to run only a single instance of the Role, *WebRolel.* It also means that our code will run on a single virtual machine in the cloud. By changing the instance count attribute from “1" to “3" (or any value), three instances of the same role will run as shown in Figure 2.9.

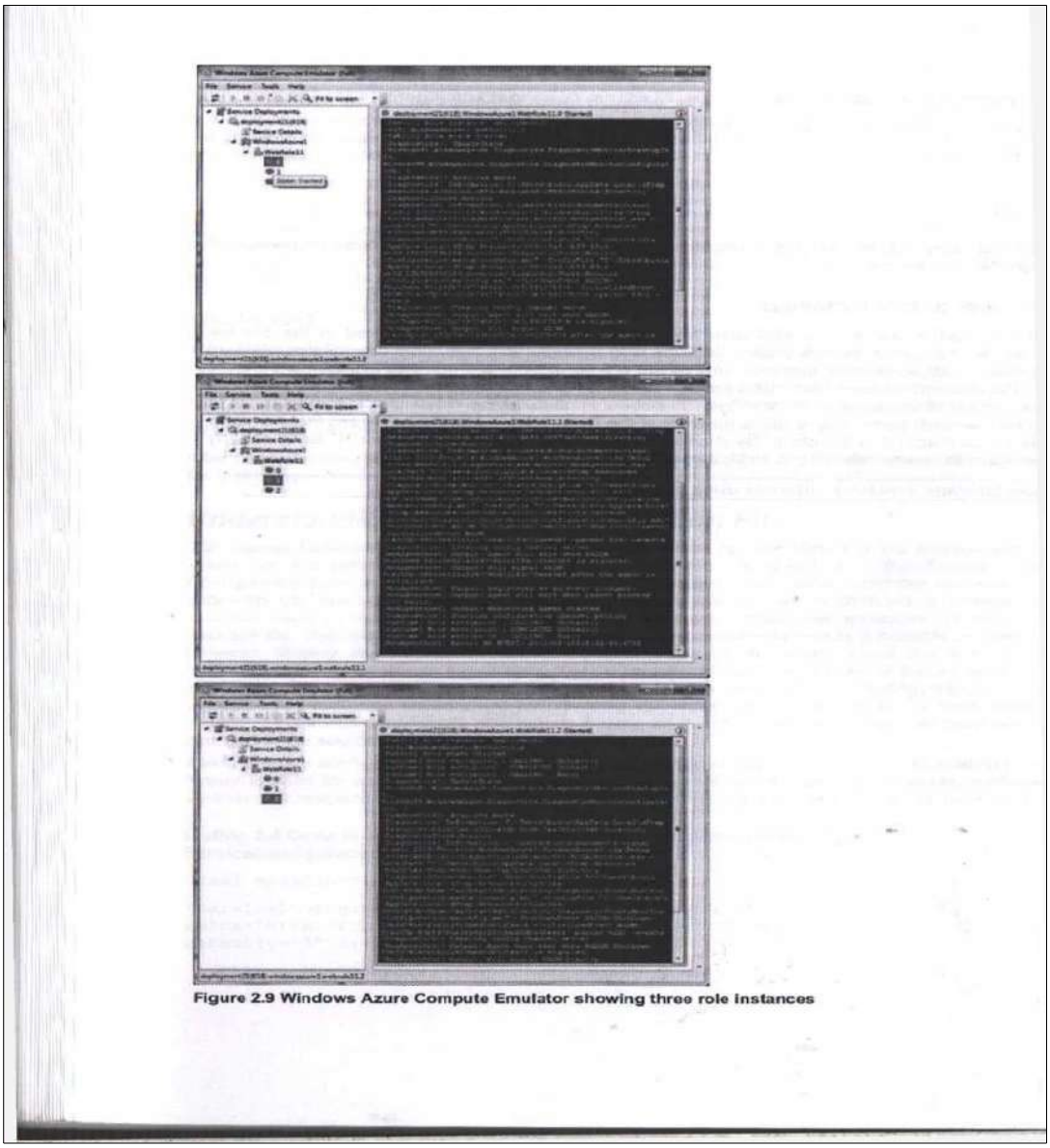


Figure 2.9 Windows Azure Compute Emulator showing three role instances

Each role instance executes in its own individual VM. On increasing the number of role instance to 3, we will have 3 VMs assigned to our application. In case of increased traffic, it is a wise idea to scale up the service by increasing the count of the role instances. That is, in order to handle additional traffic to our application, we can scale up the number of our service based on demand without redeploying our application and obviously without any infrastructure investments like servers and load-balancers too. On increasing the number of instances of roles, the incoming requests will be load balanced across the roles. Remember, we will be charged extra on each increase in the number of role instances The settings in the service configuration can be changed dynamically without need to rebuild and redeploy our application.

Note: In local environment, if we specify three instances of our website, the Emulator starts three IIS Express instances of our website and then it creates a virtual load balancer that distributes incoming request to these three applications.

Types of Roles

Roles are hence the scalable units of our application and we can increase and decrease their number of instances to scale up or down our services. There are two types of roles: *Worker* roles and *Web* roles.

- **Web Role** - It is simply a Web application running on IIS. It's able to communicate to the outside world through an HTTP or HTTPS endpoint. In general, a Web role will respond to requests, perform actions and then wait for the request.

Worker Role - It is usually meant for background processing tasks. It gets the tasks that it needs to perform through a queue where tasks in the form of messages are added by the web role(s). A worker role at its ease, fetches a message from the queue, performs the asked task and deletes the message from the queue. Note: A worker role can directly communicate with other worker roles internally over tcp and other protocols. It can also communicate with the outside world through load balancer.

Defining Configuration Settings

The *ConfigurationSettings* is an optional element where all the configuration settings for the role are defined. The configuration setting is used for specifying credential for accessing storage account services. It contains the *Setting* element to store the *name* and *value* for the configuration setting. By default, a configuration setting is predefined by *name, Microsoft.WindowsAzure.Plugins.Diagnostics.ConnectionString* with *value UseDevelopmentStorage=true* assigned to it. The purpose of this configuration setting is to define the location for storing diagnostic data. The value, *UseDevelopmentStorage=true* informs that local development storage be used for storing diagnostic information

You might be thinking, how the *web.config* file is different from the service configuration files. The main difference between *web.config* and *ServiceConfiguration.csfg* is that *web.config* is application specific whereas *ServiceConfiguration.csfg* is service specific across multiple instances and roles.

The Windows Azure storage services provide persistent, redundant storage in the cloud. The storage services include the fundamental services: Blob service, Queue service, and Table service. Before we use the actual cloud storage service, let’s see how local storage can be configured

Creating Virtual Machine

Fundamentally, there are two approaches to creating a new VM. You can upload a VM that you have built on-premises, or you can instantiate one from the pre-built images available in the Marketplace. This section focuses on the latter and defers coverage of the upload scenario until the next section.

Creating a Windows Server VM (existing portal)

To create a Windows Server VM in the management portal, complete the following steps:

1. Navigate to the management portal accessed via https://manage.windowsazure.com.
2. Click New on the command bar, and then click Compute, Virtual Machine, From Gallery.
3. Select a Windows Server image (such as Windows Server 2012 R2 Datacenter) from which to create the VM, and then click the right arrow.
4. Provide a name, tier, instance size, and administrator credentials, and then click the right arrow.
5. On the Virtual Machine Configuration screen, click Create A New Cloud Service, and then provide a DNS name, region, and storage account.
6. Leave the availability set value configured to none. (Availability sets are covered later in the chapter.) Leave the endpoints at their defaults, and click the right arrow.
7. Leave the Install VM Agent check box selected, and leave the remaining extensions cleared.
8. Click the check mark to provision the Windows Server VM.

Creating a Windows Server VM (Preview portal)

To create a Windows Server VM in the Preview portal, complete the following steps:

1. Navigate to the management portal accessed via https://portal.azure.com.
2. Click New on the command bar.
3. To navigate to the Marketplace, click the Everything link located near the upper-right corner of the blade that appears.
4. On the Marketplace blade, select Virtual Machines.
5. On the Virtual Machines blade, select Windows Server in the Recommended area.
6. On the Windows Server blade, select the image for the version of Windows Server you want for your VM (such as Windows Server 2012 R2 Datacenter).
7. Click Create.
8. On the Create VM blade, provide a host name, user name, and password for the new VM.
9. Review the Pricing Tier, Optional Configuration, Resource Group, Subscription, and Location settings and change as necessary.
10. Click Create to provision the Windows Server VM.

Creating a Linux VM (existing portal)

To create a bare bones Linux VM in the management portal, complete the following steps:

1. Navigate to the management portal accessed via https://manage.windowsazure.com.
2. Click New on the command bar, and then click Compute, Virtual Machine, From Gallery.

3. In the navigation pane, select Ubuntu.
4. Choose an Ubuntu Server image (such as Ubuntu Server 14.04 LTS) from which to create the VM, and then click the right arrow.
5. Provide a name, tier, instance size, and administrator user name.
6. Clear the Upload Compatible SSH Key For Authentication check box.
7. Select the Provide A Password check box, and provide the password for the administrator. Click the right arrow.
8. On the Virtual Machine Configuration screen, select Create A New Cloud Service, and enter a DNS name, region, and storage account.
9. Leave the availability set value configured to none. (Availability sets are covered later in the chapter.)
10. Leave the endpoints at their defaults, and click the right arrow.
11. Leave the listed extensions cleared.
12. Click the check mark to provision the Ubuntu Server Linux VM.

Creating a Linux VM (Preview portal)

To create a bare bones Linux VM in the Preview portal, complete the following steps:

1. Navigate to the management portal accessed via https://portal.azure.com.
2. Click New on the command bar.
3. To navigate to the Marketplace, click the Everything link located near the upper-right blade that appears.
4. On the Marketplace blade, select Virtual Machines.
5. On the Virtual Machines blade, select Ubuntu Server in the Recommended area.
6. On the Ubuntu Server blade, select the image for the version of Ubuntu Server you want for your VM (such as Ubuntu Server 2014.04 LTS).
7. Click Create.
8. On the Create VM blade, provide a host name and user name for the new VM.
9. Use an external tool to generate a public SSH key, and then copy and paste it in the SSH Public Key text box. See the More Info readeraid titled, “SSH key generation” for instructions on how to do this.
10. Review the Pricing Tier, Optional Configuration, Resource Group, Subscription, and Location settings and change as necessary.
11. Click Create to provision the Ubuntu Server VM.

Creating a SQL Server VM (existing portal)

The steps for creating a VM that has SQL Server installed on top of Windows Server are identical to those described earlier for provisioning a Windows Server VM using the existing portal. The only difference surfaces in the third step: instead of selecting a Windows Server image, select a SQL Server image (such as SQL Server RTM 2014 Enterprise).

Creating a SQL Server VM (Preview portal)

The steps for creating a VM that has SQL Server installed on top of Windows Server are identical to those described earlier for provisioning a Windows Server VM using the Preview

portal. The only differences surface in the fifth and sixth steps: instead of selecting a Windows Server from the Recommended area, select SQL Server from the Database Servers area, and on the SQL Server blade, select the SQL Server version you want (such as SQL Server RTM 2014 Enterprise).

Deploying Application to Windows Azure

After running and testing the application in local simulated environment provided by Compute Emulator, we can go ahead and deploy it to Windows Azure. An application has to be packaged in a format that Windows Azure understands before it is deployed. A precaution has to be taken before packaging the application. What is that? Let’s see

Commenting out the sessionState section

By default, an ASP.NET project template defines a session state provider in the Web.config file by the name *DefaultSessionStateProvider* in the sessionState section. This session state provider is configured to access the LocalDB, (an improved SQL Express instance) by default. But, when deployed to the Windows Azure, an application doesn't requires it, so ensure that the *sessionState* section in the Web.config file is commented out by enclosing it within the </— and —> tags as shown in bold in listing 2.5.

Listing 2.5 Code in Web.config file

```
<?xml version="l.0" encoding="utf-8"?>
<configuration>
<configSections>
<section name="entityFramework" type="System.Data.Entity.Internal.
ConfigFile.EntityFrameworkSection, EntityFramework, Version=6.0.0.0,
Culture=neutral, PublicKeyToken=b77a5c561934e089"
requirePermission="false" /> </configSections>
<system.diagnostics>
<trace>
<listeners>
Odd type="Microsoft.WindowsAzure.Diagnostics.
DiagnosticMonitorTraceListener, Microsoft.WindowsAzure.Diagnostics,
Version=2.1.0.0,
<filter type="" />
</add>
</listeners>
</trace>
</system.diagnostics>
<connectionStrings>
<add name="DefaultConnection" connectionString="Data Source=(LocalDb)\
vll.0;AttachDbFilename=IDataDirectory|\aspnet-WebRolell-
20130818110740.mdf; Initial Catalog=aspnet-WebRolell-
20130818110740;Integrated Security=True”
providerName="System.Data.SqlClient" />
</connectionStrings>
```

<system.web>
<compilation debug="true" targetFramework="4.5" />
•chttpRuntime targetFramework="4 . 5" />
<pages>
<namespaces>
Odd namespace="System.Web.Optimization" />
Odd namespace="Microsoft.AspNet.Identity" />
</namespaces>
<controls>
<add assembly="Microsoft.AspNet.Web.Optimization.WebForms"
namespace= "Microsoft.AspNet-Web.Optimization.WebForms"
tagPrefix="webopt" /> </controls>
</pages>
<membership>
<providers>
Cclear />
</providers>
</membership>
<profile>
<providers>
<clear />
</providers>
</profile>
<roleManager>
<providers>
<clear />
</providers>
</roleManager>
<!—<sessionState mode="InProc"
customProvider="DefaultSessionProvider"> <providers>
<add name="DefaultSessionProvider" type="System.Web.Providers.
DefaultSessionStateProvider, System.Web.Providers,
Version=l.0.0.0,
Culture=neutral, PublicKeyToken=3lbf3 85 6ad3 64e3 5"
connectionstringName="DefaultConnection" />
</providers>
</sessionState>—>
</system.web>
<entityFramework>
<defaultConnectionFactory type="System.Data.Entity.Infrastructure.
LocalDbConnectionFactory, EntityFramework">
<parameters>
<parameter value="vll.0" />

```
</parameters>
</defaultConnectionFactory>
<providers>
<provider invariantName="System.Data.SqlClient"
type="System.Data.Entity.SqlServer.
SqlProviderServices, EntityFramework.SqlServer" />
</providers>
</entityFramework>
</configuration>
```

Now , the application can be packaged for deployment.

Packaging the application

To make an application work in the cloud environment, it has to be first packaged in a format that Windows Azure is able to identify. To create a *service package* of our application, right click on our project, *WindowsAzurel* in the *Solution Explorer* and select *Package* option. The *Package Windows Azure Application* dialog will appear as shown in Figure 2.10.

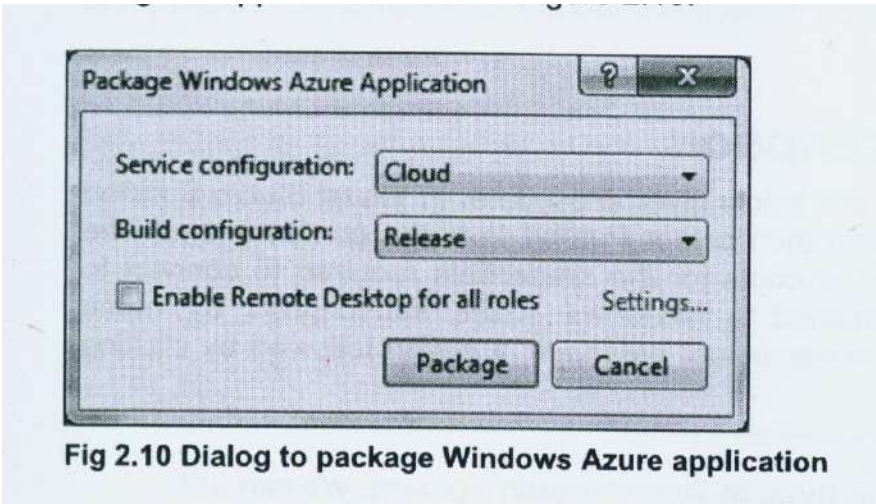


Fig 2.10 Dialog to package Windows Azure application

The *Service configuration* drop-down displays two options, *Local* and *Cloud* allowing us to generate *Cloud* as well as *Local* configuration files, *ServiceConfiguration.Cloud.cscfg* and *ServiceConfiguration.Local.cscfg.* Since, this package will be deployed to Windows Azure, we will select the *Cloud* option to generate the Cloud configuration file.

The *Build Configuration* drop down helps in choosing the option that help in determining the type of configuration file that we wish to create. The two available options are *Debug* and *Release* which indicates that the two configuration files are meant for different operations:

- Debug configuration - Used while debugging an application on a local machine
- Release configuration - Used while building an application for deployment

Since, we wish to use the configuration file for deploying the application, we will select the *Release* option. Finally, select the *Package* button to create the service package and configuration files. Visual Studio will open a folder that contains two files, *ServiceConfiguration.Cloud.cscfg* and *WindowsAzurel.cspkg files* as shown in Figure 2.11. The .cspkg extension is for service package files. A service package file contains the service definition as well as the binaries and other items for the application being deployed.

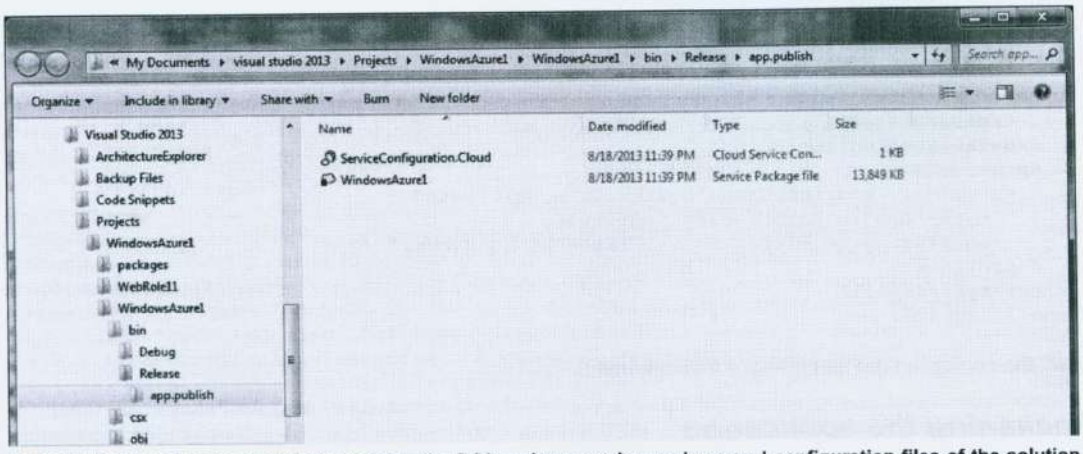


Fig 2.11 Windows Explorer window showing the folder where service package and configuration files of the solution are created

Connecting to Windows Azure Subscription

To deploy the application to Windows Azure, we will follow the below given procedure. In Visual Studio Solution Explorer window, right-click the project, *WindowsAzure1* and then select *Publish* from the context menu. The *"Publish Windows Azure Application"* dialog opens up that prompts for the credentials required to connect to Windows Azure subscription. I assume, you have obtained a Windows Azure subscription by name *Subscription1.* From the "Choose your subscription" drop down, select *<Manage..>* option followed by clicking Import button.

Note: To create Windows Azure subscription, visit the URL,

https://account.windowsazure.com/Home/Index and sign in through your Microsoft account. We can use default subscriptions, "Azure Free Trial", "Pay-As-You-Go" and can even create a new subscription from the different plans that are displayed.

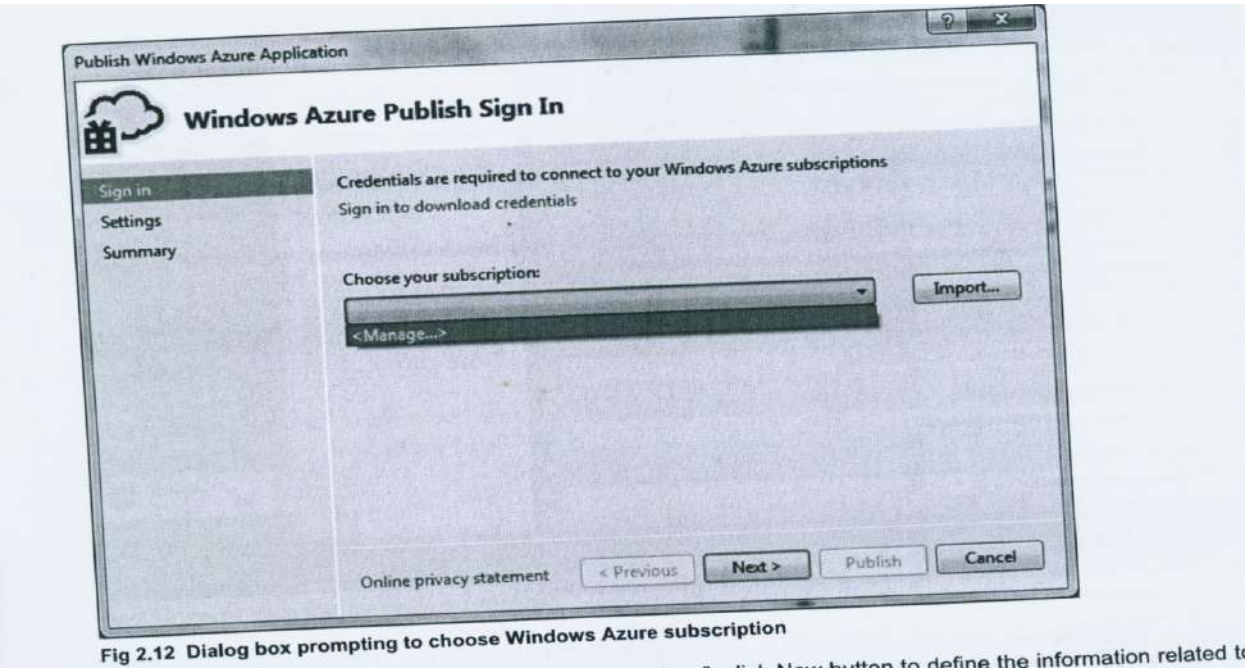


Fig 2.12 Dialog box prompting to choose Windows Azure subscription

In the next dialog, "Manage Windows Azure Subscriptions", click New button to define the information related to Windows Azure subscription (to which we want to upload our Visual Studio application). The New Subscription dialog appears as shown in Figure 2.13 that displays different options that enable Visual Studio to manage and deploy our application to our Windows Azure account

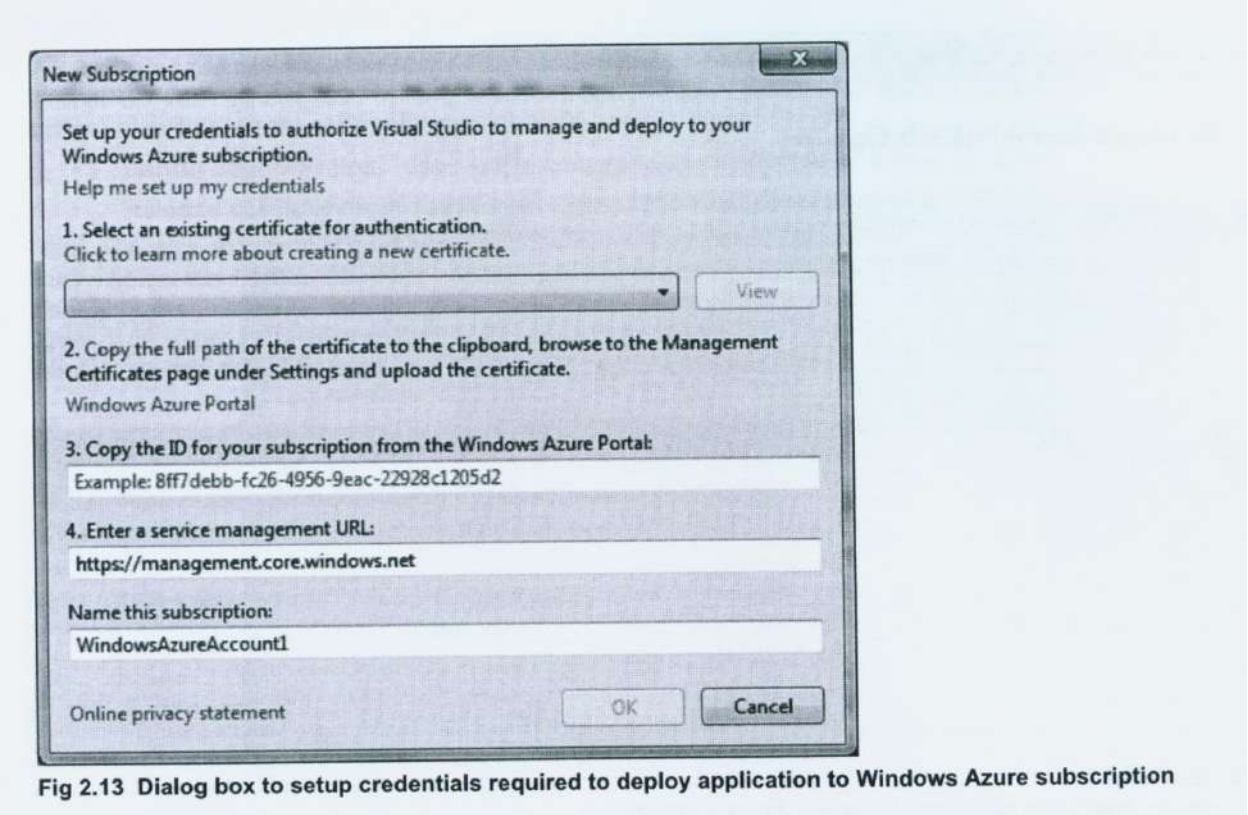


Fig 2.13 Dialog box to setup credentials required to deploy application to Windows Azure subscription

We will be using the certificate to authenticate or establish connection between Visual Studio and our Windows Azure subscription. After selecting the existing certificate, we will copy its full path to the clipboard as it will be required while uploading the certificate to Windows Azure portal.

Fig 2.14 Portal Page that manages certificates

From the Windows Azure portal, click *Settings* option followed by clicking *Management Certificates* tab. A message appears informing that no management certificate exists (see Figure 21.4). Click, "*Upload Management Certificate*" link to upload the certificate. A dialog box appears as shown in Figure 2.15 to specify the management certificate and the Windows Azure subscription to which we want to deploy our application. Click the *File* button to browse the drive and select the management certificate that we want to upload to Windows Azure portal. From the *Subscription* drop down select the Windows Azure subscription, *Subscription* to which we want to deploy our application. Click the right button to initiate uploading of certificate to Windows Azure portal.

Fig 2.15 Dialog box for uploading the management certificate

Once successfully uploaded, the management certificate will appear on the Windows Azure portal. Copy the ID of the Windows Azure subscription in the *New Subscription* dialog. Also assign a name, *WindowsAzureAccountI* to this new subscription (see Figure 2.13) and finally click *OK* button to create it. The newly created subscription, *WindowsAzureAccountI* will automatically appear in the *"Publish Windows Azure Application"* dialog as shown in Figure 2.16. Click *Next* button to move further.

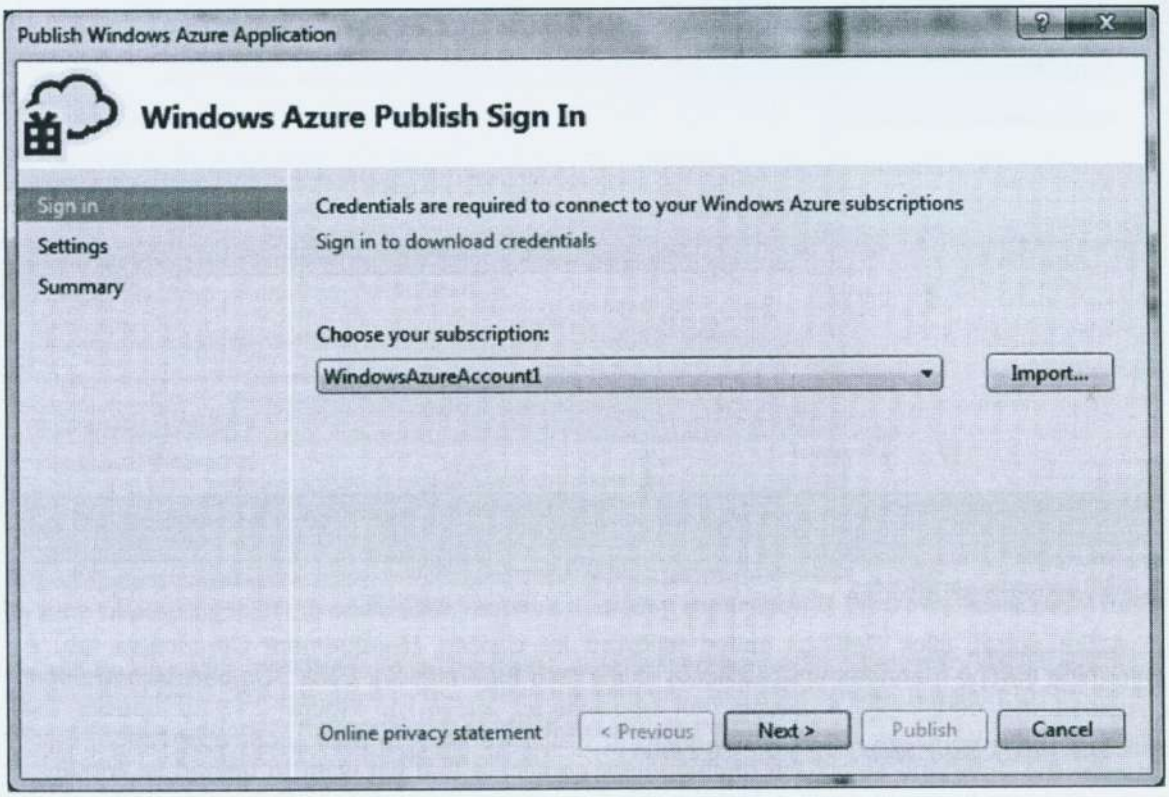


Fig 2.16 Dialog box for selecting the subscription to deploy Visual Studio application

The next dialog prompts for common publish settings as shown in Figure 2.17. From the *Cloud Service* dropdown, we select an existing cloud service or even create a new one. Let us create a cloud service called *bmharwani* in *East Asia* region through Windows Azure portal. A cloud service is a container for running our package in the cloud. It also provides a public URL to access our web roles, and enable us to upload packages in two environments, staging and production environment.

Note: The region plays a major role in a Cloud service. These regions represent the locations where Microsoft has its data centers. Usually, the region that is closest to the consumers of our service is selected to avoid network latency.

The staging environment is used for testing our application and the production environment is for making our tested application available to the consumers of our application. From the *Environment* box, we select the environment i.e. whether we want to deploy our application to production or staging environment.

Note: An application can be deployed to either a staging or a production environment within the same Cloud Service. Traditionally, an application is deployed first to the staging environment and then, it is upgraded to the production environment. The only difference in the two environments is in the URL that we use to access them. In the staging environment the URL to access the web role will be like *http://bmharwani7329pqr409.doudapp.net*,while in the production environment, the URL is very friendly like, http://bmharwani.cloudapp.net

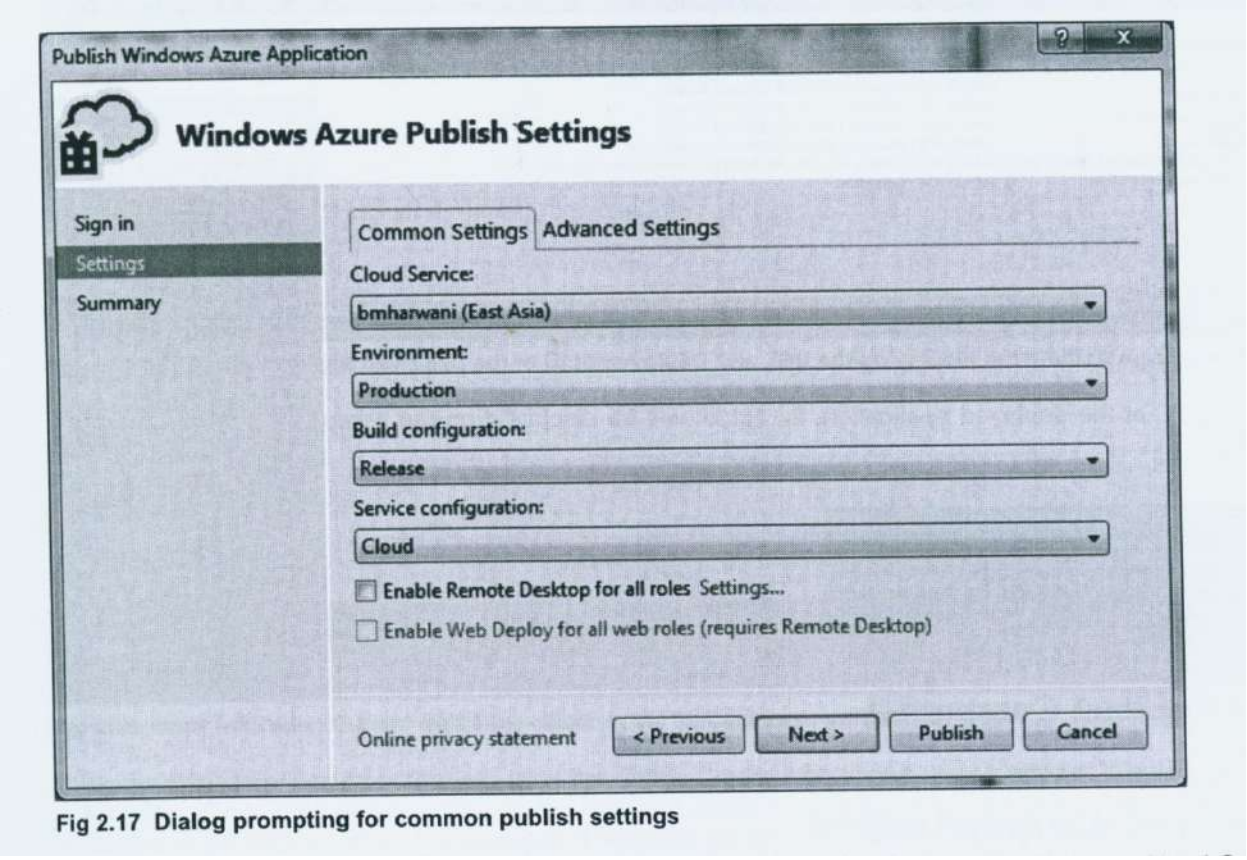


Fig 2.17 Dialog prompting for common publish settings

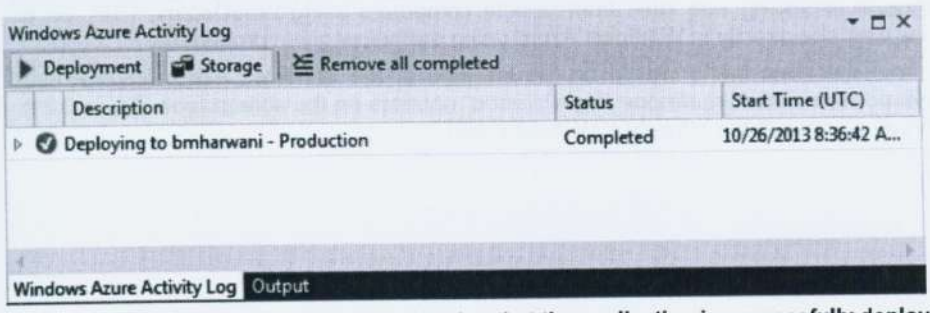


Fig 2.18 Windows Azure Activity log indicating that the application is successfully deployed.

The Visual Studio will initiate the procedure to deploy our Visual Studio application to the *Cloud Service* called *bmharwani.* Once deployed, its status and the start time will be displayed as shown in Figure 2.18.

On visiting the URL of the deployed application, its output will be displayed on the screen as shown in Figure 2.20**.**

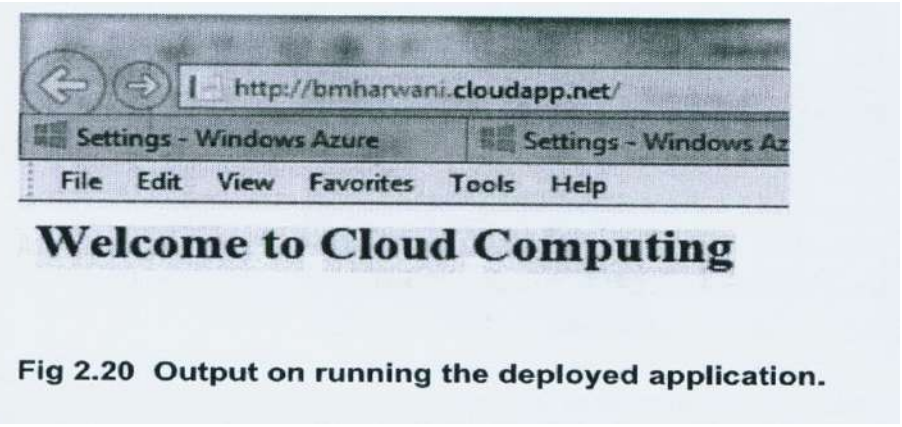


Fig 2.20 Output on running the deployed application.

Storing data

The Windows Azure Platform provides Windows Azure Storage as well as SQL Azure for storing data. SQL Azure is for storing the relational databases of the application. Azure Storage though does not support relation data but is popular for its scalability and cost advantages.

Using Azure Storage Services

Azure Storage Services consist of highly scalable and available persistent storage for the following three types of data:

Tables are structured tabular data stored in an Entity-Attribute-Value (EAV) data model; the maximum size of all attribute values of an entity is 1MB. Entities can be grouped into storage partitions.

- Blobs consist of unstructured file-based data stored in an array of bytes. A blob container contains blobs of up to 50GB in size can be stored in hierarchical groups. Only blob containers and their content are available for public access.

Queues contain an unlimited number of messages stored in tables for processing by global services; messages have a maximum size of 8KB. Messages usually are automatically deleted from the queue when it is read by the respective process.

Note: To assure availability and reliability, all stored data is replicated to multiple data centers to get the data back in the event of a data center's destruction. Also, the total capacity of a storage account for all blob, table, and queue data is set to 200TB.

SQL Azure Database (SADB, formerly SQL Data Services, SDS, and SQL Server Data Services, SSDS) is an alternative to Azure Tables that offers many features of relational tables.

Types of Storage

Windows Azure supports three types of storage:

- **Blob storage** - Used for storing large sized data.
- **Table storage** - Provides structured storage for applications. It is stored in the cloud environment hence is not relational data storage.
- **Queue storage** - Provides asynchronous transfer of data between different parts of an application.

Blob Storage

The simplest way to store data in Windows Azure storage is to use blobs. A storage account can have one or more containers, each of which holds one or more blobs. Blobs can be divided into blocks for efficient transmission. If a failure occurs in transmission, the recent block is re-transmitted instead of sending the whole blob again. Blobs can also have associated metadata for keeping additional information. Blobs are unstructured; to work with data in a structured format Windows Azure storage provides tables.

Table Storage

Azure Table is the structured storage and supports scalable tables in the cloud. Remember, these are not relational tables and contain data in a set to entities with properties. It can store billions of entities holding terabytes of data. The Azure table storage supports the Language Integrated Query (LINQ) extensions for .NET 3.0 or later versions, ADO.NET Data Services, and representational state transfer (REST), which allows applications developed using non-.NET languages to access table storage via the Internet.

To access Azure storage we need to create an Azure storage account. When a new Azure storage account is created, a 256-bit public shared key will be sent to us which is used while initializing storage table access. The access will be authenticated based on the secret key. Authentication is done for each request to table storage through the secret key. When REST is used to access table storage, the account name is part of the host name in the URL string. The URL is constructed with the format of http://<accountName>.table.core.windows.net.

The following are the key parts of the table storage specification:

- **Table:** A table contains a set of entities. Table names are associated to the account. Any number of tables can be created for an application within a storage account.

Entity: An entity is a row in cloud table storage and is a set of properties.

Property: A table has no defined schema but properties. Properties are values stored in an entity of various types, such as int, string, Bool, etc. The name for a property is case-sensitive. A property can have values of types as listed in table 3.1.

Table 3.1 A brief description of Property types and their values

Property Type	**Details**
Binary	An array of bytes up to 64 KB
Bool	A Boolean value
DateTime	A 64-bit value expressed as UTC time; range is 1/1/1600 to 12/31/9999
Double	A 64-bit floating point value
GUID	A 128-bit globally unique identifier
Int	A 32-bit integer
Int64	A 64-bit integer
String	A UTF-16-encoded value; may be up to 64 KB

- **PartitionKey:** Every table has a special property called PartitionKey. Since the actual data of table storage is physically distributed to many storage nodes, which may cross many storage servers running in the cloud, the cloud storage system uses this key to manage the storage nodes distribution.

RowKey: RowKey is the unique ID of the entity, and an entity can be identified using the combination of the PartitionKey and RowKey in a table.

Timestamp: The Timestamp indicates the time a table is created.

- **Partition:** The Partition is a logical set of entities defined in a table with the same PartitionKey value.
- **Sort Order:** A single index is provided for all entities in a table. Data entities are sorted by PartitionKey and then RowKey. This makes queries specifying these keys more efficient.

Queue Storage

The purpose of Queue storage is not on storing data but to enable communication between Web role and Worker role instances. The Web role instance that receives this request can write a message into a queue describing the work to be done. A Worker role instance that's waiting on this queue can then read the message and carry out the task it specifies. Any results can be returned via another queue.

Windows Azure storage can be accessed either by a Windows Azure application or by an application running somewhere else. In both cases, all three Windows Azure storage styles use the conventions of REST (and the OData protocol for tables) to identify and expose data. Everything is named using URIs and accessed with standard HTTP operations, so clients can be created using .NET, Java, or other familiar technologies.

The Windows Azure platform charges independently for compute and storage resources. This means that an onpremises application can use just Windows Azure storage, accessing its data in the RESTful way just described. For example, a Windows Server application running in an enterprise data center might choose to store backups in Windows storage blobs.

Using Blob Storage

Blob stands for 'binary large object and is just a sequence of bytes. One Windows Azure Storage Account can have multiple Containers. A Container can be considered as a collection of one or more Blobs. Also, each Blob can have one or more metadata properties to provide additional information about the blob. The metadata properties are Name-Value collection of strings. The contents of every Blob on Windows Azure can be accessed by browsing its corresponding URI. That is, the Azure storage is based on REST. All storage objects are available through standard HTTP calls to a named URI.

Windows Azure Blob enables applications to store large objects, up to 50GB each in the cloud. The system is highly available and durable. We can always access your data from anywhere at any time, and the data is replicated at least 3 times for durability.

Each storage account has access to blob storage. For each account there can be 0..n containers. Each container contains the actual blobs (raw array of bytes). Containers can be public or private. In public containers, the URLs to the blobs can be accessed over the internet while in a private container, only the account holder can access those blob URLs.

Configuring Local Storage

Local storage is a temporary storage area provided by the operating system where data related to an application is stored for future use. It is not a physical storage but a logical storage that our role instance uses to store and retrieve data locally. This local storage area is non sharable and is not shared among other role instances. Also, the storage is of volatile nature and its content are lost if something goes wrong to our VM. To configure local storage, we need to specify the *name* of the storage, its *capacity* in mega bytes and whether we wish to delete its data when the role is recycled. The following example, configures a volatile local storage by name *LocalStorage* of size *100 MB:*

<WebRole name="WebRolel">

<LocalResources>

<LocalStorage name="<MyLocalStorage" cleanOnRoleRecycle="true" sizeInMB="100" />

</LocalResources>

</WebRole>

Note: The maximum size of local storage that we can use for a single role instance is 20 GB.

The *name* attribute defines the name of the storage and *cleanOnRoleRecycle attribute* is set to value *true* to indicate that it is a volatile local storage i.e. its data will be lost when the role is recycled, its new version is deployed or if server crashes. The *sizeInMB* attribute defines the size of the local storage as 100 MB.

The local storage once configured can be accessed in a role. To access local storage in a role, we will be making use of the RoleEnvironment class defined in the ServiceRuntime assembly. Let's know more about the ServiceRuntime assembly and its RoleEnvironment class.

Using ServiceRuntime Assembly

The ServiceRuntime assembly performs co-ordination between the Windows Azure runtime and the service. A service interacts with the Windows Azure through this assembly. The assembly provides several APIs and methods that are very helpful in getting run time information of a service. Like, we can use this assembly to know the running status of our service in the cloud, the number of running roles, the number of instances of each role, different configuration settings, and much more.

Note: The ServiceRuntime assembly is automatically referenced in our application on creating a new ASP.NET web role in Visual Studio.

To fetch different information about the running service, we make use of the RoleEnvironment class in the ServiceRuntme assembly. Following are the few of the properties, methods and events of RoleEnvironment class that are used frequently to know the status of the running service:

GetConfigurationSettingValue(string) - Retrieves configuration settings and the latest value from ServiceConfiguration.cscfg file.

GetLocalResource(string) - Gets the local path of the local resource **RequestRecycle()** - Recycles the role instance.

DeploymentId - Returns a unique ID assigned to our running service.

Roles - Returns a collection of all the roles in our running service and their instances.

Following call is used to request information about the local storage resource.

LocalResource MyLocalStorage =RoleEnvironment.GetLocalResource(

"MyLocalStorage");

The RoleEnvironment.GetLocalResource() method is called passing in the name of our local storage, *MyLocalStorage.* The object returned by *GetLocalResource()* method includes three properties, *Name, MaximumSizelnMegabytes,* and *RootPath.* As the name suggests, the *Name* and *MaximumSizelnMegabytes* properties return the *name* and *capacity* of the local storage and the *RootPath* property returns the physical path of the folder where the local storage area is created. It is using the *RootPath* property that data is written and retrieved from the local storage folder.

Creating and Accessing Local Storage

We know by now that RoleEnvironment class of the ServiceRuntime assembly is the one that is used for accessing and using local storage as it informs the starting place to store the data. The Windows Azure compute emulator tool simulates the local storage feature on our machine enabling us to test, debug and confirm if our application is able to use the local storage or not before actually deploying it the cloud.

The steps to create and access a local storage are as given below:

1. Create a new Windows Azure Cloud Service by the name say, *DemoLocalStorage.*
2. From the *New Windows Azure Cloud Service* dialog that pops up, add an *ASP.NET Web Application* to the Windows Azure Cloud Service solution. Leave its name set to the default, *WebRolell.*

Select Web Forms template from the next dialog followed by clicking the Create Project button.

3. Two projects, DemoLocalStorage and WebRolel 1 will be created. To configure the web role, WebRolel 1 to create and access local storage, open the Solution Explorer window, in the DemoLocalStorage project, under *Roles,* double-click the *WebRolell* node to open the *Role Configuration* dialog.
4. Click the *Local Storage* tab in the left pane and, from the right pane, click the *Add Local Storage* button. A new local storage by default name, *LocalStoragel* will be added

Name this new local storage as *MyLocalStorage* and type *50* into the *Size* text box to indicate that we wish to have a local storage of size 50 MB.

5. Do not check the *Clean On role recycle* check box (refer Figure 3.1) as we wish the local storage to be of non volatile nature. That is, we don't want to loose the content of the local storage when the role is recycled.

Save the WebRolell configuration.

On saving the web role's configuration, the Visual Studio integrated development environment (IDE) automatically modifies the *ServiceDefinition.csdef* file to reflect the actions taken in the visual *Role Configuration* dialog. The code of the Service Definition file will appear as shown in Listing 3.1.

Listing 3.1 Code in Service Definition file

```
<?xml version="l.0" encoding="utf-8"?>
<ServiceDefinition name="DemoLocalStorage"
xmlns="http://schemas.microsoft.com/ServiceHosting/2008/10/ServiceDefinition"
schemaVersion="2013-03.2 . 0">
<WebRole name="WebRolell" vmsize="Small">
<Sites>
<Site name="Web">
<Bindings>
<Binding name="Endpointl" endpointName="Endpointl" />
</Bindings>
</Site>
</Sites>
<Endpoints>
clnputEndpoint name=''Endpointl" protocol="http" port="80" />
</Endpoints>
<Imports>
<Import moduleName="Diagnostics" />
</Imports>

<LocalResources>
<LocalStorage name="MyLocalStorage" cleanOnRoleRecycle="false" sizeInMB="50" />
</LocalResources>
</WebRole>
</ServiceDefinition>
```

As expected, an element <LocalStorage> is added in the <LocalResources> element confirming the creation of a local storage. *The name, cleanOnRoleRecycle* and *sizelnMB* attributes of the *<LocalStorage>* element are set to values, *MyLocalStorage, false* and *50* respectively, hence reflecting the values set by us in the *Role Configuration* dialog.

On deploying the service on Windows Azure or in the simulated environment of *Compute Emulator,* the service requests for the required disk space from the Windows Azure fabric. The Windows Azure fabric uses the configuration file for determining the size and type of local storage to create. The *<LocalStorage>* element defined inside the *<LocalResources>* element guides the fabric to create a virtual machine with at least *50 MB* of free disk space and makes it available by the name *MyLocalStorage.*

The local resources including the local storage is accessed through different methods of the *RoleEnvironment* class. We have discussed that by passing the resource name in the form of string parameter to the *GetLocalResource()* method of the *RoleEnvironment* class , we get a *LocalResource* object. The LocalResource object has several properties like *Name, MaximumSizeInMegabytes* and the *RootPath* that returns the *name, capacity* and the physical path of the folder where the local storage area is created.

Design the web form to access and display the properties of the newly created local storage. The Default.aspx web form of the web role, WebRolell contains certain default controls as shown in Figure 3.2(top). Delete all the default controls on the web form and for the purpose of displaying the configuration values of the local storage, drag and drop a *Label* control on the web form. The web form, Default.aspx with a Label control will appear as shown in Figure 3.2(bottom)

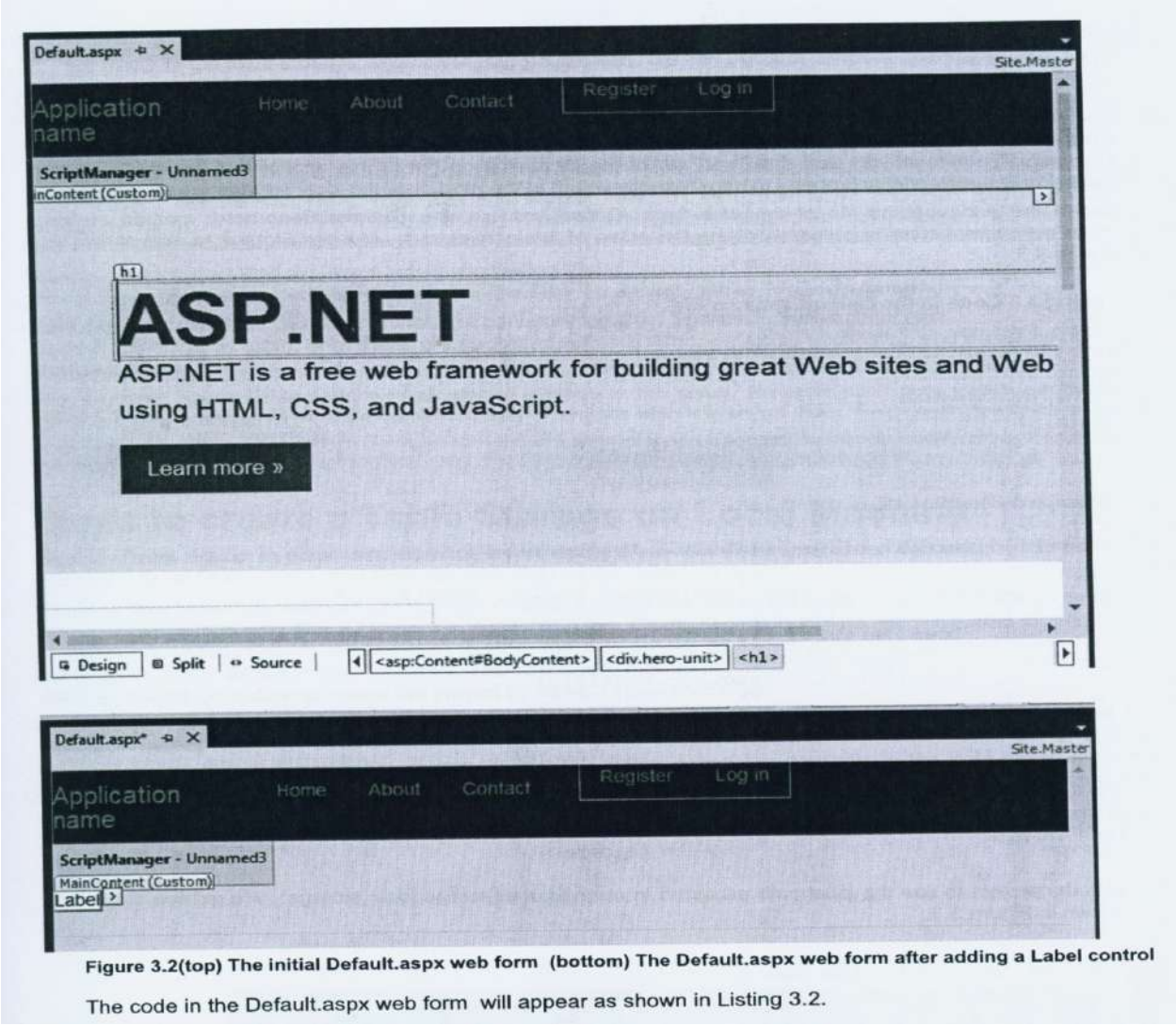


Figure 3.2(top) The initial Default.aspx web form (bottom) The Default.aspx web form after adding a Label control

The code in the Default.aspx web form will appear as shown in Listing 3.2.

in the Default.aspx web form will appear as shown in Listing 3.2.

Listing 3.2 Code in the Default.aspx web form

```
<%@ Page Title="Home Page" Language="C#" MasterPageFile="~/Site.master"
AutoEventWireup="true" CodeBehind="Default.aspx.cs"
Inherits="WebRolell._Default" %>
<asp:Content ID="BodyContent" runat="server">
<br />
<asp:Label ID="Labell" runat="server" Text="Label"x/asp:Label> <br />
</asp:Content>
```

9. Next step is to access the local storage and display its properties via the Label control dropped in the web form, Default.aspx. Though, we can access and display all the three properties, *Name, MaximumSizelnMegabytes* and the *RootPath* of the local storage but in this application, we will focus to display only the *RootPath* property to know exactly where in the hard disk, the local storage area is created.

So, in the code behind file of the web form, Default.aspx.es, the *GetLocalResourcef)* method of the *RoleEnvironment* class is called supplying the name of the local storage, *MyLocalStorage* to it as shown in Listing 3.3.

Listing 3.3 Code in the Default.aspx.cs file

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using Microsoft.WindowsAzure.ServiceRuntime;
namespace WebRolel {
public partial class _Default : System.Web.UI.Page {
protected void Page_Load(object sender, EventArgs e)
{
LocalResource resource = RoleEnvironment.GetLocalResource( "MyLocalStorage");
Labell.Text = resource.RootPath;
}
}
}
```

The *GetLocalResource()* method returns a *LocalResource* object, *resource.* The RootPath property of the returned LocalResource object is then assigned to the Label control to display the local path assigned to the Local storage.

10. Run the project to see the local path assigned to our MyLocalStorage local storage. We get the output as shown in Figure 3.3.

Figure 3.3 Application showing the path of the local storage

In the output, we can see that the local storage path where the Windows Azure Compute Emulator deploys the service is displayed. The long set of alphabets and numbers after Resources informs the sequence number of the deployment on the machine. The path also includes a folder, *directory* inside which a folder by the name of the local storage, *MyLocalStorage* is created. We can perform different operations with the local storage using this path.

Using Windows Azure Table Service

The Compute service is also referred as Hosted Service. It is the application we deploy on Windows Azure. Every Hosted Service can have Web role and Worker role. A Web role is an ASP.NET Web application accessible via an HTTP or HTTPS endpoint and is commonly the front-end for an application. A worker role is a role that is useful for generalized development, and may perform background processing for a Web role.

We can develop the Windows Azure application using Visual Studio and Windows Azure SDK. Also, we can deploy the application to local Windows Azure Emulator for testing, and to Windows Azure.

A Windows Azure Cloud Service includes two configuration files: *ServiceDefinition.csdef* and *ServiceConfiguration.cscfg.* These files are packaged with our Windows Azure application and deployed to Windows Azure.

The Windows Azure Table service is structured storage in the cloud. An application may create many tables within a storage account. A table contains a set of entities also known as rows. Each entity contains a set of properties. An entity can have at most 255 properties including the mandatory system properties - PartitionKey, RowKey, and Timestamp. "PartitionKey" and "RowKey" form the unique key for the entity.

Steps to create a Table Storage on Local Machine

Launch Visual Studio in administrator mode by selecting Start->AII Programs option followed by right-clicking Microsoft Visual Studio 2013, and then selecting Run as administrator option.

Create a new project by selecting *File->New Project.* Expand the *Visual C# node* from the Installed Templates list and select *Cloud* template. Choose the *Windows Azure Cloud Service* template, set the *Name* of the project to *TableServiceApp.* Specify the *Location* to store the new project by selecting *Browse* button. Let the *Solution name* be the same as the project name i.e. *TableServiceApp.* Keeping the *Create directory for solution* checkbox checked, select OK button to create the project by name *TableServiceApp.*

In the New Windows Azure Cloud Service dialog, from the left pane, expand the node for the Visual C# language, select *ASP.NET Web Application* from the list of available roles and click the *arrow (>)* to add an instance of this role to the solution. Before closing the dialog, select the new role in the right panel, click the *pencil* icon and rename the role to *ProductWebRole.* We can also rename the role by right clicking on it and typing a new name. Finally select *OK* button (see Figure 3.4).

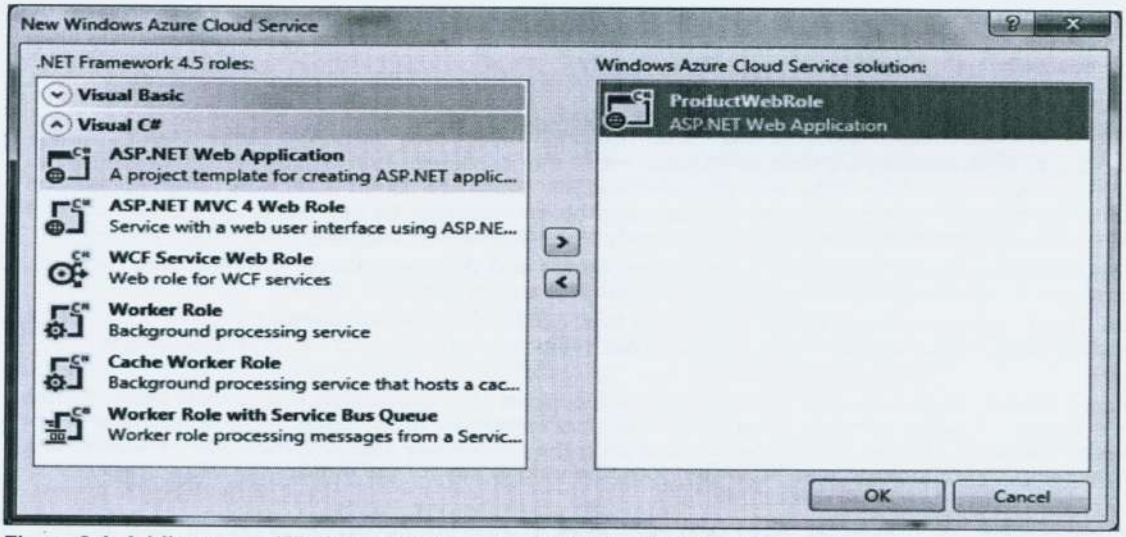


Figure 3.4 Adding a new Windows Azure Cloud Service to the solution

From the next dialog, select *Web Forms* template followed by clicking *Create Project* button to generate the cloud service solution. When the solution is created, Visual Studio opens the *Default.aspx* page in the code window. In the Solution Explorer pane, we will see that two projects are created, *TableServiceApp,* a cloud service project and the *ProductWebRole,* an ASP.NET Web Application project. Under the *TableServiceApp* cloud service project, we will find one definition and two configuration files. The definition file, *ServiceDefinition.csdef* defines the runtime settings for the application including the required roles, endpoints, etc. There are two configuration files:

ServiceConfiguration.Local.cscfg - We can configure this local configuration file to use the local Windows Azure Storage emulator for testing the application locally

- *ServiceConfiguration.Cloud.cscfg* - We can configure this file to use Windows Azure storage service.

Note: If we find only see one .cscfg file, we need to install the latest Windows Azure Tools for Microsoft Visual Studio.

We will perform the following steps to create our application :

1. Creating the Data Model project and Defining an Entity class
2. Providing storage account settings
3. Creating a data source class to enable access to the Table Storage data.
4. Creating a Web Role to take information of new product

Creating the Data Model Project

Our application will store information of the products using Windows Azure Table service. In the Table service, the information is stored as a collection of entities. Entities are similar to rows. An entity has a primary key and a set of properties where properties represent columns of a table and are stored as name/value pair. So, for storing information of the products, we will define an entity consisting of certain properties like, product_code, product_name etc. In addition to the properties, every entity in Table service has two key properties: the *PartitionKey* and the *RowKey* which together form the table's primary key and uniquely identify each entity in the table. We will be creating an Entity class for defining our product entity.

After defining entity, we will create a data source so that the ASP.NET Web Application, *ProductWebRole* that we just created can use the data source to access the Table service. The data source will contain the methods for storing and retrieving products in the storage.

Note: To access the Table service, we can use LINQ.

For creating Entity class, we first need to create a data model project. To create a data model project, from the Solution Explorer, right-click our solution, *TableServiceApp,* select Add->New Project option. In the *Add New Project* dialog, expand the *Visual* C# node from the *Installed Templates.* Select *Windows* category followed by choosing *Class Library* project template from the right pane. Assign the name as ProductInfo to our project and select *OK* button as shown in Figure 3.5

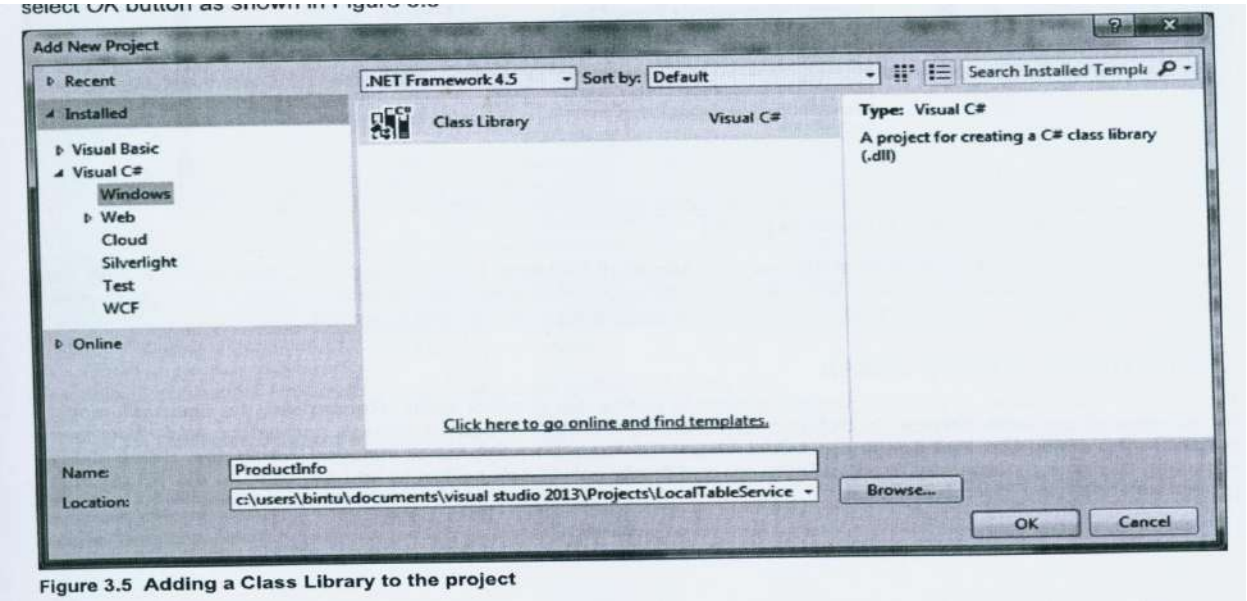


Figure 3.5 Adding a Class Library to the project

Delete the default class file, *Class1.cs* that is generated by the class library template. In the Solution Explorer window, inside the ProductInfo project, right-click *Class1.cs* and choose *Delete.* Click *OK* in the confirmation dialog. Next, we need to add a reference to the .NET Client Library for WCF Data Services in the ProductInfo project. Right-click the ProductInfo project, and then click *Add Reference.* In the *Add Reference* dialog, switch to the Assemblies->Framework tab and select *System.Data.Services.Client.* From *Assemblies->Extensions* tab, select *Microsoft.WindowsAzure.StorageClient* and *Microsoft.WindowsAzure.ServiceRuntime* and then click OK . (Fig 3.6)

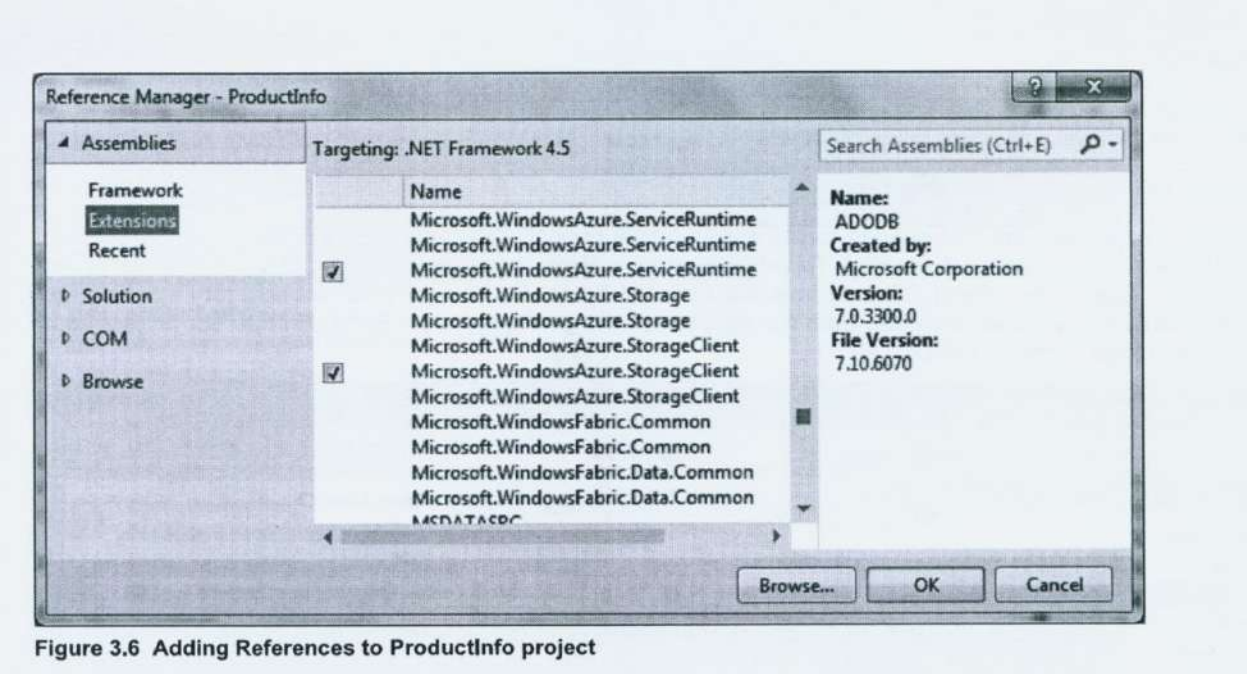


Figure 3.6 Adding References to ProductInfo project

The references, *System.Data.Services.Client* and *Microsoft. Windows.StorageClient* are used for accessing the Windows Azure storage services. The reference, *Microsoft.WindowsAzure.ServiceRuntime* is an environment and runtime API that is used for retrieving data connection string in the configuration file.

Defining an Entity class

We need to add a class to our *ProductInfo* project to define our product entity. Remember, the class will model the schema of our table service. In Solution Explorer, right-click *ProductInfo* project, and select *Add->New Item* option. In the dialog box that opens up, select *Visual C# Items* from the list of *Installed Templates.* Select *Class* template, assign it a name say *ProductEntity.cs* and finally select *Add* button to add the class to our *ProductInfo* project as shown in Figure 3.7

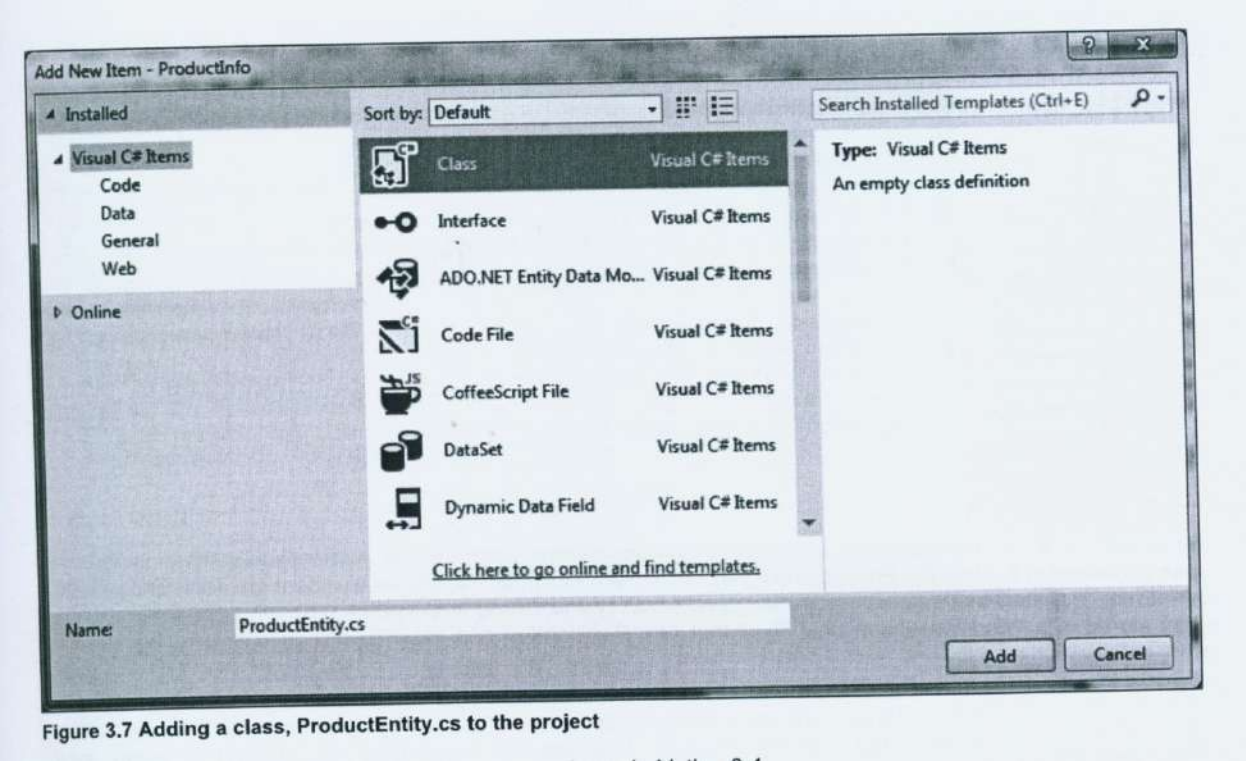


Figure 3.7 Adding a class, ProductEntity.cs to the project

The default content of *ProductEntity.cs* class is as shown in Listing 3.4.

Listing 3.4 Default code in ProductEntity.cs file

```
using System;

using System.Collections.Generic; using System.Linq; using System.Text; using System.Threading.Tasks;

namespace ProductInfo {

class ProductEntity {
```

Modify the ProductEntity.cs class to appear as shown in Listing 3.5.

Listing 3.5 Code written in ProductEntity.cs file

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Microsoft.WindowsAzure.StorageClient;          #1
using Microsoft.WindowsAzure.Storage.Table;
```

```
namespace ProductInfo {
public class ProductEntity : TableEntity {
public ProductEntity()
{
PartitionKey = DateTime.UtcNow.ToString("MMddyyyy");
RowKey = string.Format("{0:10}_{1}", DateTime.MaxValue.Ticks - DateTime.Now.Ticks, Guid.NewGuid());
}
public int Product_Code { get; set; } public string Product_Name { get; set; } public string Product_Category { get; set; ) public int Product_Quantity { get; set; } public double Product_Price { get; set; )
}
}
```

The statement, #1 is inserted to use the *Microsoft. WindowsAzure.StorageClient* namespace so as to import the types contained in it. Also, the ProductEntity class is declared as public and is set to inherit the TableEntity class. TableEntity is a class that defines the *PartitionKey, RowKey a*nd *TimeStamp* system properties required by every entity stored in a Windows Azure table. Together, the *PartitionKey* and *RowKey* are used to uniquely identify every entity within a table.

A default constructor is also added to the *ProductEntity* class to initialize its *PartitionKey* and *RowKey* properties. The date on which the product is added to the service is used as the *PartitionKey* which means that there will be a separate partition for each day of product entries. The value of the partition key is used to ensure load balancing of the data across storage nodes. Tables within partitions are sorted in *RowKey* order. The products will be displayed in sorted order with the latest entry shown at the top. Since, we will be storing code, name, quantity, category and price of the products, we add properties for *Product_Code, Product_Name, Product_Category, Product_Quantity* and *Product_Price* in the class. Save the schema class, ProductEntity.cs file.

Note: The Timestamp system property of the entity in the Table service allows the service to keep track of when an entity was last modified. This Timestamp field is intended for system use and should not be accessed by the application.

Let's move on to the next step to provide storage account settings.

Providing Storage Account Settings

Next, we need to provide our storage account settings. To create a new setting follow the below given steps:

1. In Solution Explorer, under *TableServiceApp* project, expand *Roles* node and then double-click *ProductWebRole* to open the properties dialog for this role.
2. Select the Settings tab in the dialog that opens up. We will find a default setting, Microsoft.WindowsAzure. Plugins. Diagnostics. ConnectionString.

To add a new setting, select *Add Setting* button, Enter the name of Setting as *DataConnectionString* (any name). From the *Type* drop down list, select the *Connection String* option. In the Value box, select the ellipses button (...). We get the dialog box titled, *Create Storage Connection String.* Select the option,

Connecting using: *Windows Azure storage emulator* followed by selecting *OK* button as shown in Figure 3.8. It is for sure that to access Windows Azure Blob, Queue, and Table services, we need to create a storage account in the Management Portal but for the purpose of local testing, we can use the Windows Azure storage emulator, which is included in the Windows Azure SDK development environment that simulates the Blob, Queue, and Table services available in the cloud.

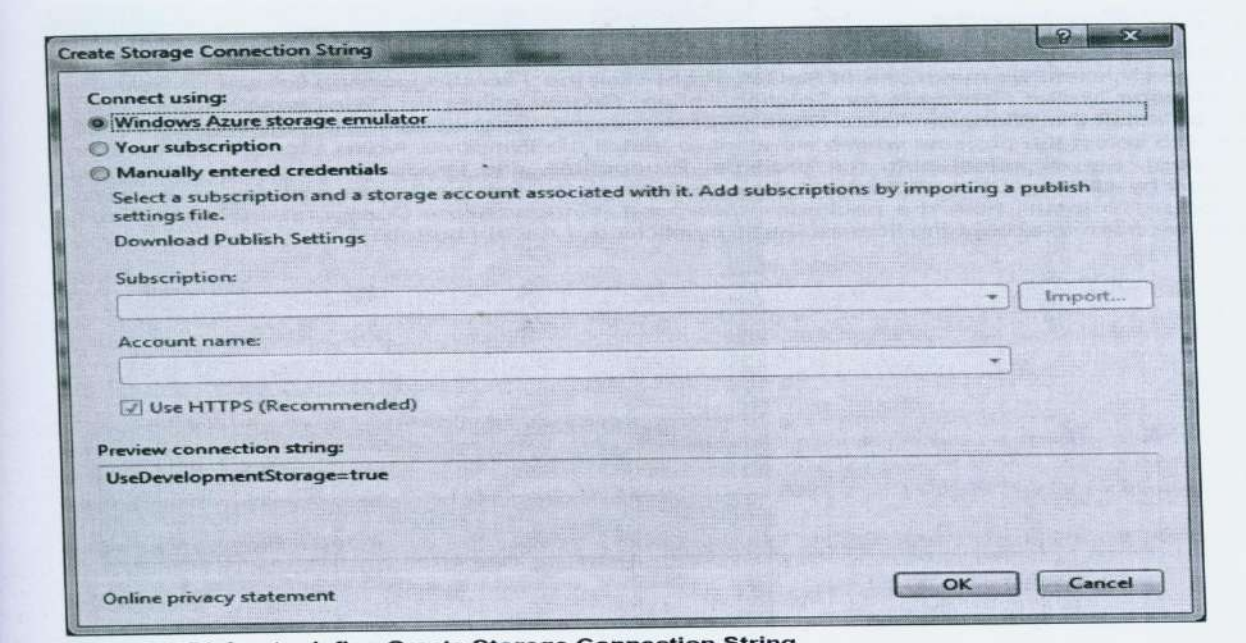


Figure 3.8 Dialog to define Create Storage Connection String

The value for the new setting, *DataConnectionString* will be set as *UseDevelopmentStorage keyword-true* as shown in Figure 3.9. The Development storage means our local computer where the Azure fabric is installed. The newly created connection string will be stored in our application's service configuration files.

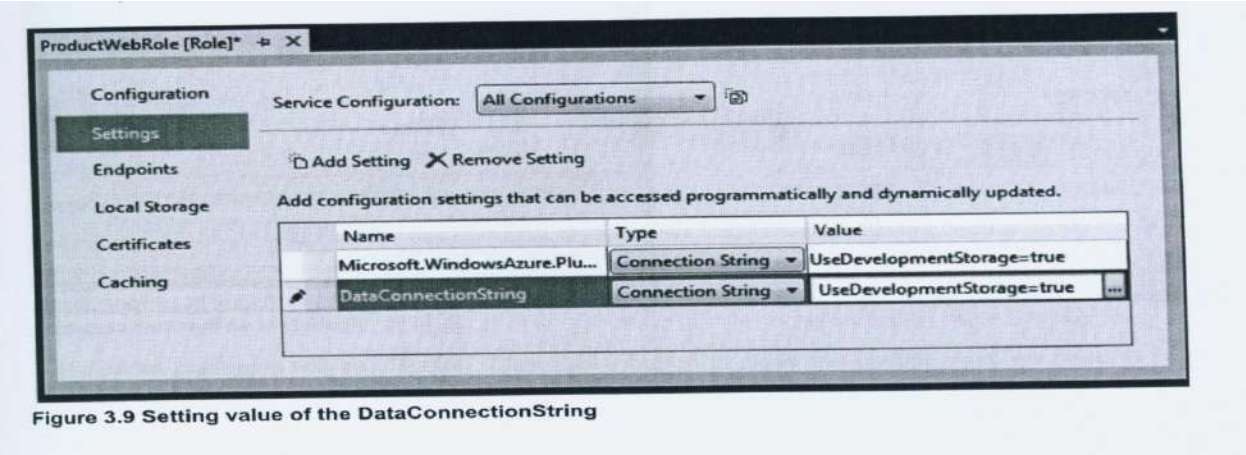


Figure 3.9 Setting value of the DataConnectionString

Steps to create virtual machine using windows azure cloud.

Step 1 − Login to Azure Management Portal.

Step 2 − Locate and click on 'Virtual Machines' in the left panel and then click on 'Create a Virtual Machine'.

Step 3 − Alternatively, click 'New' at the bottom left corner and then click 'Compute' → 'Virtual Machine' →'Quick Create'.

Step 4 − Enter DNS name. This has to be unique. The DNS name is used to connect to the virtual machine.

Step 5 − Select the image and size from the dropdown list. The size affects the cost of running virtual machine.

Step 6 − Enter username and password. You must remember to log in to the virtual machine later.

Step 7 − Select the relevant region.

Step 8 − Click on 'Create a virtual machine' and you are ready to use your new machine. It will take a few seconds for the machine to be created.

Q.Give the various types of storage services in windows azure.[W-16](7M)[S-18]

Ans:

Azure Storage services

Azure Storage includes these data services:

- Azure Blobs: A massively scalable object store for text and binary data.
- Azure Files: Managed file shares for cloud or on-premises deployments.
- Azure Queues: A messaging store for reliable messaging between application components.
- Azure Tables: A NoSQL store for schemaless storage of structured data.

Blob storage

Azure Blob storage is Microsoft's object storage solution for the cloud. Blob storage is optimized for storing massive amounts of unstructured data, such as text or binary data.

Blob storage is ideal for:

- Serving images or documents directly to a browser.
- Storing files for distributed access.
- Streaming video and audio.
- Storing data for backup and restore, disaster recovery, and archiving.
- Storing data for analysis by an on-premises or Azure-hosted service.

Objects in Blob storage can be accessed from anywhere in the world via HTTP or HTTPS. Users or client applications can access blobs via URLs, the Azure Storage REST API, Azure PowerShell, Azure CLI, or an Azure Storage client library. The storage client libraries are available for multiple languages, including .NET, Java, Node.js, Python, PHP, and Ruby.

Azure Files

Azure Files enables you to set up highly available network file shares that can be accessed by using the standard Server Message Block (SMB) protocol. That means that multiple VMs can share the same files with both read and write access. You can also read the files using the REST interface or the storage client libraries.

One thing that distinguishes Azure Files from files on a corporate file share is that you can access the files from anywhere in the world using a URL that points to the file and includes a shared access signature (SAS) token. You can generate SAS tokens; they allow specific access to a private asset for a specific amount of time.

File shares can be used for many common scenarios:

- Many on-premises applications use file shares. This feature makes it easier to migrate those applications that share data to Azure. If you mount the file share to the same drive letter that the on-premises application uses, the part of your application that accesses the file share should work with minimal, if any, changes.
- Configuration files can be stored on a file share and accessed from multiple VMs. Tools and utilities used by multiple developers in a group can be stored on a file share, ensuring that everybody can find them, and that they use the same version.
- Diagnostic logs, metrics, and crash dumps are just three examples of data that can be written to a file share and processed or analyzed later.

At this time, Active Directory-based authentication and access control lists (ACLs) are not supported, but they will be at some time in the future. The storage account credentials are used to provide authentication for access to the file share. This means anybody with the share mounted will have full read/write access to the share.

Queue storage

The Azure Queue service is used to store and retrieve messages. Queue messages can be up to 64 KB in size, and a queue can contain millions of messages. Queues are generally used to store lists of messages to be processed asynchronously.For example, say you want your customers to be able to upload pictures, and you want to create thumbnails for each picture. You could have your customer wait for you to create the thumbnails while uploading the pictures. An alternative would be to use a queue. When the customer finishes his upload, write a message to the queue. Then have an Azure Function retrieve the message from the queue and create the thumbnails. Each of the parts of this processing can be scaled separately, giving you more control when tuning it for your usage.

Table storage

Azure Table storage is now part of Azure Cosmos DB. To see Azure Table storage documentation, see the Azure Table Storage Overview. In addition to the existing Azure Table storage service, there is a new Azure Cosmos DB Table API offering that provides throughput-optimized tables, global distribution, and automatic secondary indexes. To learn more and try out the new premium experience, please check out Azure Cosmos DB Table API.

Deploy application to windows azure cloud

Create and deploy

1. Log in to the Azure portal.
2. Click **Create a resource > Compute**, and then scroll down to and click **Cloud Service**.
3. In the new **Cloud Service** pane, enter a value for the **DNS name**.
4. Create a new **Resource Group** or select an existing one.
5. Select a **Location**.
6. Click **Package**. This opens the **Upload a package** pane. Fill in the required fields. If any of your roles contain a single instance, ensure **Deploy even if one or more roles contain a single instance** is selected.
7. Make sure that **Start deployment** is selected.
8. Click **OK** which will close the **Upload a package** pane.
9. If you do not have any certificates to add, click **Create**.

Verify your deployment completed successfully

1. Click the cloud service instance.

   The status should show that the service is **Running**.

2. Under **Essentials**, click the **Site URL** to open your cloud service in a web browser.

Explain how azure maximizes data availability and minimizes security risk

Maximizing data availability:

WA provides robust availability based on extensive redundancy achieved with visualization:

a. **Replicated data**

Azure provides failover clustering of thrice replicated data and Hosted application instances running when failure occurs.

b. **Geographically distributed data**:

Customers can leverage the geographically distributed nature if WA infrastructure by creating a second storage account to provide hot-failover capability. Customers may also write customized roles to extract data from storage for offsite private backup.

Minimizing security risks:

a. Secure socket layer transmission encryption for web roles:

Azure services can enable Transport layer security (TLS) to use secure HTTP protocol (HTTPS) for transmission of encrypted requests to and responses from production Hosted services and storage accounts for web roles.

b. Encrypting information in Azure storage services:

NET 3.5 provides implementations of many standard cryptographic algorithms including symmetric (shared secret key) and asymmetrical (Public Key Infrastructure PKI)

c. Azure's conformance with SAS70 and ISO/IEC 27001:2005 certifications.

i. Statement on Auditing Standards no. 70 (SAS 70)

It includes the service auditor's opinion on the fairness of presentation of the service organizations description of controls that had been placed in operation and the suitability of the design of the controls to achieve the specified control objectives.

ii. The ISO/IEC 27001:2005 standard

An ISO/IEC 27001 complaint system will provide a systematic approach to ensuring the availability, confidentiality and integrity of corporate information. Using controls based on identifying and combating the entire range of potential risks to the organizations information assets.

Discuss development facilities in azure cloud

Windows Azure is an open, flexible, and robust cloud computing platform. It enables you to quickly build, deploy, and manage applications in a cloud environment across a global network of Microsoft-managed, high-performance datacenters.

Windows Azure reduces your up-front infrastructure costs and improves business efficiency. It is arguably the fastest way for your product to hit the market and scale elastically on demand.

Windows Azure makes it easy for you to leverage the power of cloud computing from where you are. You can host your existing applications on Windows Azure, or extend your on-premise applications to have a cloud interface, or build new cloud applications on Windows Azure from the ground up.

You can use the language, tool or framework of your choice for building applications. Here are some of the things you can do with Windows Azure:

- Web Sites: Build and deploy bespoke websites (or open source CMS) to a highly scalable cloud environment integrated with flexible Windows Azure services.
- Cloud Services: Create highly-available, infinitely scalable applications and services using the technology of your choice.
- Mobile Services: Shorten your app's time to market by adding a scalable and secure backend for development tasks like storage, authentication, and push notification.
- Data Store: Store just data on Windows Azure, with applications using this data running on-premises (outside the public cloud).

Capabilities such as the above combined with virtual machines, data management (SQL Database, tables, blobs), business analytics (Hadoop, SQL reporting), messaging, caching, identity, media, commerce, and high-performance computing services can satisfy any of your application needs.

The four Azure Services Platform components—Windows Azure, .NET Services, SQL Services, and Live Services—offer these rich technical capabilities:

- **Windows Azure provides** scalable computation and storage to user applications and to other Azure Services Platform components.
- **DOTNET Services** coordinate user login credentials across differing security schemes and offers distributed infrastructure services to cloud-based and local applications.
- **SQL Services** provide data storage for applications running in the cloud and in corporate data centres.
- **Live Services** allows users to coordinate data across all of their devices and selectively share data with friends and associates.